USENIX

LISA XVIII: 18th Large Installation System Administration Conference

Atlanta, GA, Nov. 14–19, 2004

conference

proceedings

LISA XVIII

Eighteenth Large Installation System Administration Conference

Atlanta, Georgia, USA

November 14–19, 2004

Sponsored by

The USENIX Association and **SAGE**

USENIX

SAGE

The People Who Make IT Work

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: +1 510-528-8649
http://www.usenix.org
<office@usenix.org>

Past LISA Conferences

Large Installation Systems Admin. I Workshop	1987	Phildelphia, PA
Large Installation Systems Admin. II Workshop	1988	Monterey, CA
Large Installation Systems Admin. III Workshop	1989	Austin, TX
Large Installation Systems Admin. IV Conference	1990	Colorado Spgs, CO
Large Installation Systems Admin. V Conference	1991	San Diego, CA
Systems Administration VI Conference	1992	Long Beach, CA
Systems Administration VII Conference	1993	Monterey, CA
Systems Administration VIII Conference	1994	San Diego, CA
Systems Administration IX Conference	1995	Monterey, CA
Systems Administration X Conference	1996	Chicago, IL
Systems Administration XI Conference	1997	San Diego, CA
Systems Administration XII Conference	1998	Boston, MA
Systems Administration XIII Conference	1999	Seattle, WA
Systems Administration XIV Conference	2000	New Orleans, LA
Systems Administration XV Conference	2001	San Diego, CA
Systems Administration XVI Conference	2002	Philadelphia, PA
Systems Administration XVII Conference	2003	San Diego, CA

ISBN 1-931971-24-2

USENIX Association

Proceedings of the Eighteenth Large Installation System Administration Conference (LISA XVIII)

November 14–19, 2004
Atlanta, GA, USA

ACKNOWLEDGMENTS

PROGRAM CHAIR
Lee Damon, *University of Washington*

PROGRAM COMMITTEE & SHEPHERDS
David Blank-Edelman, *Northeastern*
Rudi van Drunen, *Leiden Pathology and Cytology Labs*
Esther Filderman, *Pittsburgh Supercomputing Center*
Jon Finke, *RPI*
Æleen Frisch, *Exponential Consulting*
Michael Gilfix, *IBM*
David Hoffman, *Stanford University*
Brendan Murphy, *Microsoft Research*
Mario Obejas, *Raytheon*
John Sechrest, *Public Electronic Access to Knowledge*
Jeff Sheldon, *Louisiana State University*
Snoopy, *Infocopter Entertainment GmbH*

SHEPHERD
Alva Couch, *Tufts University*

SCRIBE
Ian Masterson, *University of Washington*

INVITED TALKS COORDINATORS
Chair: Deeann M. M. Mikula, *Consultant*
Adam S. Moskowitz, *Menlo Computing*
Marcus Ranum, *Security Consultant*

GURU IS IN COORDINATOR
Philip Kizer, *Texas A&M University*

WORK-IN-PROGRESS COORDINATOR
Snoopy, *Infocopter Entertainment GmbH*

WEB RECORDING
Esther Filderman, *Pittsburgh Supercomputing Center*

TRAINING PROGRAM COORDINATOR
Daniel V. Klein, *USENIX Association*

WORKSHOP COORDINATOR
Gretchen Phillips

CONFIGURATION WORKSHOP
Paul Anderson, *University of Edinburgh*

SYSADMIN EDUCATION WORKSHOP
Curt Freeland, *University of Notre Dame*
John Sechrest, *PEAK Internet Services*

AFS WORKSHOP
Esther Filderman, *Pittsburgh Supercomputing Center*
Derrick Brashear, *Carnegie Mellon University*

ADVANCED TOPICS WORKSHOP
Adam S. Moskowitz, *Menlo Computing*

USENIX BOARD LIAISON
Jon "maddog" Hall, *Linux International*

SAGE EXEC LIAISON
Gus Hartmann

LOCAL COLOR
William LeFebvre, *CNN Internet Technologies*

PROCEEDINGS TYPESETTING
Rob Kolstad, *SAGE*

USENIX ASSOCIATION
Cat Allman, *Sales and Marketing Director*
Marci Chase, *Conference Manager*
Jennifer Joost, *Conference Manager*
Anne Dickison, *Marketing Communications Manager*
Jane-Ellen Long, *IS/Production Director*
Tony Del Porto, *System Administrator*
Ellie Young, *Executive Director*

REFEREED PAPER EXTERNAL REVIEWERS

Derrick J Brashear, *OpenAFS*
Alva Couch, *Tufts University*
Christophe Kalt, *Tykhe Capital LLC*
Jason Heiss, *Overture, a Yahoo! company*
Cat Okita, *Earthworks*
John "Rowan" Littell, *Earlham College*
Mary Seabrook, *TellMe*
Steve Shah, *NetScaler*
Brendan Strejcek, *the University of Chicago*
Steve VanDevender, *University of Oregon*
Yi-Min Wang, *Microsoft Research*
Kenytt Avery, *Willing Minds LLC*
Fuat Baran, *Micromuse, Inc.*
Mark Boltz, *Stonesoft Inc.*
Terrell Countryman, *Georgia Institute of Technology*
Alan S. Epps, *Nielsen Media Research*
Benjy Feen, *Monkeybagel.com*
Steve Hanson, *University of Wisconsin, River Falls*
David Harnick-Shapiro, *M9 Systems, Inc*
Ray W. Hiltbrand, *E*Trade Financial*
Doug Hughes, *Global Crossing Inc.*
Mark Lamourine, *Mercury Computer Systems*
William LeFebvre, *CNN Internet Technologies*
Mark R. Lindsey, *Engineers' Consulting Group of Valdosta, Georgia, USA*
Michael Matthews, *America Online*
Shane B. Milburn, *Intel*
Ellen Mitchell, *Texas A&M University*
Joe Morris, *Turner Broadcasting*
Will Partain, *Verilab*
Dustin Puryear, *Puryear Information Technology, LLC*
Gene Rackow, *Argonne*
Rob Siemborski, *Carnegie Mellon University*
Jennifer Sturm, *Hamilton College*
Garry Zacheiss, *MIT IS&T Server Operations*
Alexander Zangerl, *Bond University, Australia*
Ted Cabeen, *Impulse Internet Services*

CONTENTS

Acknowledgments ii

Author Index vi

Message from the Program Chair vii

Opening Remarks

Wednesday, 8:45 a.m.–9:00 a.m. **Chair: Lee Damon**

Keynote

Wednesday, 9:00 a.m.–10:30 a.m. **Speaker: Howard Ginsberg, CNN**

SPAM/Email

Wednesday, 11:00 a.m.–12:30 p.m. **Rudi Van Drunen**

Scalable Centralized Bayesian Spam Mitigation with Bogofilter 1
Jeremy Blosser and David Josephsen, VHA, Inc.

DIGIMIMIR: A Tool for Rapid Situation Analysis of Helpdesk and Support E-mail 21
Nils Einar Eide, Andreas N. Blaafadt, Baard H. Rehn Johansen, and Frode Eika Sandnes, Oslo University College

Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management 33
Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson, Microsoft Research; Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo, National Taiwan University

Intrusion and Vulnerability Detection

Wednesday, 2:00 p.m.–3:30 p.m. **Yi-Min Wang**

A Machine-Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing 47
Sufatrio, Temasek Laboratories, National University of Singapore; Roland H. C. Yap, School of Computing, National University of Singapore; Liming Zhong, Quantiq International

DigSig: Run-time Authentication of Binaries at Kernel Level 59
Axelle Apvrille, Trusted Logic; David Gordon, Ericsson; Serge Hallyn, IBM LTC; Makan Pourzandi and Vincent Roy, Ericsson

I^3FS: An In-Kernel Integrity Checker and Intrusion Detection File System 67
Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok, Stony Brook University

Configuration Management

Wednesday, 4:00 p.m.–5:30 p.m. — Mike Gilfix

Nix: A Safe and Policy-Free System for Software Deployment 79
Eelco Dolstra, Merijn de Jonge, and Eelco Visser, Utrecht University

Auto-configuration by File Construction: Configuration Management with Newfig 93
William LeFebvre and David Snyder, CNN Internet Technologies

AIS: A Fast, Disk Space Efficient "Adaptable Installation System" Supporting Multitudes of Diverse Software Configurations 105
Sergei Mikhailov and Jonathan Stanton, George Washington University

Networking

Thursday, 9:00 a.m.–10:30 a.m. — Jon Finke

autoMAC: A Tool for Automating Network Moves, Adds, and Changes 113
Christopher J. Tengi, Princeton University; James M. Roberts, Tufts University; Joseph R. Crouthamel, Chris M. Miller, and Christopher M. Sanchez, Princeton University

More Netflow Tools: For Performance and Security 121
Carrie Gates, Michael Collins, Michael Duggan, Andrew Kompanek, and Mark Thomas, Carnegie Mellon University

Monitoring and Troubleshooting

Thursday, 11:00 a.m.–12:30 p.m. — Mike Gilfix

Real-time Log File Analysis Using the Simple Event Correlator (SEC) 133
John P. Rouillard, University of Massachusetts at Boston

Combining High Level Symptom Descriptions and Low Level State Information for Configuration Fault Diagnosis 151
Ni Lao, Tsinghua University; Ji-Rong Wen and Wei-Ying Ma, Microsoft Research Asia; Yi-Min Wang, Microsoft Research

System Integrity

Thursday, 2:00 p.m.–3:30 p.m. — John Sechrest

LifeBoat – An Autonomic Backup and Restore Solution 159
Ted Bonkenburg, Dejan Diklic, Benjamin Reed, and Mark Smith, IBM Almaden Research Center; Michael Vanover, IBM PCD; Steve Welch and Roger Williams, IBM Almaden Research Center

PatchMaker: A Physical Network Patch Manager Tool 171
Joseph R. Crouthamel, James M. Roberts, Christopher M. Sanchez, and Christopher J. Tengi, Princeton University

Who Moved My Data? A Backup Tracking System for Dynamic Workstation Environments 177
Gregory Pluta, Larry Brumbaugh, William Yurcik, and Joseph Tucek, NCSA, University of Illinois

Security

Friday, 9:00 a.m.–10:30 a.m. **David Hoffman**

Making a Game of Network Security 187
Marc Dougherty, Northeastern University

Securing the PlanetLab Distributed Testbed 195
Paul Brett, Mic Bowman, Jeff Sedayao, Robert Adams, Rob Knauerhase, and Aaron Klingaman, Intel Corporation

Secure Automation: Achieving Least Privilege with SSH, Sudo and Setuid 203
Robert A. Napier, Cisco Systems

Theory

Friday, 11:00 a.m.–12:30 p.m. **John Sechrest**

Experience in Implementing an HTTP Service Closure 213
Steven Schwartzberg, BBN Technologies; Alva Couch, Tufts University

Meta Change Queue: Tracking Changes to People, Places, and Things 231
Jon Finke, Rensselaer Polytechnic Institute

Solaris Zones: Operating System Support for Consolidating Commercial Workloads 241
Daniel Price and Andrew Tucker, Sun Microsystems, Inc.

AUTHOR INDEX

Robert Adams 195
Axelle Apvrille 59
Andreas N. Blaafadt 21
Jeremy Blosser 1
Ted Bonkenburg 159
Mic Bowman 195
Paul Brett 195
Larry Brumbaugh 177
Michael Collins 121
Alva Couch 213
Joseph R. Crouthamel 113, 171
Dejan Diklic 159
Eelco Dolstra 79
Marc Dougherty 187
Michael Duggan 121
Nils Einar Eide 21
Jon Finke 231
Carrie Gates 121
David Gordon 59
Serge Hallyn 59
Yennun Huang 33
Baard H. Rehn Johansen 21
Aaron Johnson 33
Merijn de Jonge 79
David Josephsen 1
Anand Kashyap 67
Aaron Klingaman 195
Rob Knauerhase 195
Andrew Kompanek 121
Sy-Yen Kuo 33
Ni Lao 151
William LeFebvre 93
Wei-Ying Ma 151
Sergei Mikhailov 105
Chris M. Miller 113
Robert A. Napier 203
Swapnil Patil 67
Gregory Pluta 177
Makan Pourzandi 59
Daniel Price 241
Benjamin Reed 159
James M. Roberts 113, 171
John P. Rouillard 133
Roussi Roussev 33
Vincent Roy 59
Christopher M. Sanchez 113, 171
Frode Eika Sandnes 21
Steven Schwartzberg 213
Jeff Sedayao 195
Gopalan Sivathanu 67
Mark Smith 159
David Snyder 93
Jonathan Stanton 105
Sufatrio 47
Christopher J. Tengi 113, 171
Mark Thomas 121
Joseph Tucek 177
Andrew Tucker 241
Michael Vanover 159
Chad Verbowski 33
Eelco Visser 79
Yi-Min Wang 33, 151
Steve Welch 159
Ji-Rong Wen 151
Roger Williams 159
Ming-Wei Wu 33
Roland H. C. Yap 47
William Yurcik 177
Erez Zadok 67
Liming Zhong 47

Message from the Program Chair

Welcome to the 18th LISA Conference. Thank you for attending.

The Authors, Program Committee, Track Chairs, Readers, and Reviewers have done tremendous work to put together a very strong conference. I hope you are well rested; it will be a busy and thrilling week for everyone.

The theme of this year's conference – "System Administration Reality – Automation, Configuration, and Users" – is meant to focus our attention on the day-to-day issues of our profession while at the same time providing an opportunity to look towards the future of System Administration and our very active role in its development.

Perpetual problems such as backups and printing don't appear to be getting any closer to A Solution than they were when this conference started 18 years ago. However, many other problems have been resolved or have reasonable solutions proposed for them. These pages contain some of those proposed solutions as well as solutions for other problems you may not have encountered yet.

It is the combination of research and day-to-day System Administration that is the strength of this conference and its community. We all contribute to the development of System Administration as a profession. We all contribute to the tools and tricks of this trade. We all have much to be proud of in the work represented in these Proceedings as well as that of the previous 17.

If you hanker for System Administration as a discipline (or profession) with specialized information, you need look no further than this tome to see the sort of specialization that is emerging.

These Proceedings from the 2004 LISA Conference contain the complete texts of the refereed papers. Of the 70 papers submitted, 22 are presented here. It is always challenging for a Program Committee to select papers. Many good papers were not accepted due to lack of room or for other reasons. Every such decision is made with regret and a hope for resubmission to future conferences.

I would like to thank everyone who submitted an abstract for consideration. I would especially like to thank the authors who have put so much effort into writing these papers. Further, I would like to thank all of the people who worked so hard to make this conference happen.

All of us hope that you find these works useful and enjoyable and that you consider presenting your own work at future LISA conferences. LISA 2005 is December 4–9th in San Diego, California. It is never too early to start writing your paper abstracts.

Lee Damon
Program Chair
LISA 2004

Scalable Centralized Bayesian Spam Mitigation with Bogofilter

Jeremy Blosser and David Josephsen – VHA, Inc.

ABSTRACT

Bayesian content filters gained popular acclaim when they were put forward in 2002 by Paul Graham as a potential long-term solution for the spam problem. They have since fallen from the limelight, however, due to perceived attack vulnerabilities inherent to all content-based filters as well as real and imagined vulnerabilities specific to Bayesian filters. It has also been assumed that Bayesian filters would be problematic to implement in centralized or large environments due to wordlist management issues. This paper revisits the effectiveness of Bayesian filters as a sustainable singular spam solution for mid- to large-sized environments through a real-world study of the deployment and operation of the Bogofilter Robinson-Fisher Bayesian classification utility in a production mail environment servicing thousands of accounts. Our implementation strategy and methodology as well as our results are described in detail so that they can be evaluated and replicated if desired. Other filtering methodologies which were previously implemented in this environment are also discussed for comparison purposes, though they have since been removed from production due primarily to lack of need. Bayesian classification has been able to solve the spam problem for this user population for the present and observable future, with a single wordlist, and with no secondary spam filtering techniques employed. Significantly, only two business-related legitimate messages have been reported as blocked due to filter misclassification since Bogofilter was deployed.

Introduction

Unsolicited bulk and commercial email, popularly known as spam, is one of the most critical issues facing systems, mail, and network administrators today. More and more human and system resources are being dedicated both to dealing with the day-to-day filtering of mail and to determining any possible long-term solutions to the problem, be they technical, social, or legislative. Whatever solutions are applied, however, are inevitably subverted or defeated by spammers within a short time of implementation, causing many to characterize the situation as an arms race. Fixed string content filters are avoided by mangling commonly blocked words. MTA blacklists cause spammers simply to change their mail routes and service providers and cause excessive collateral damage [JAC]. Challenge-response systems have led to spammers adding mail route harvesting to their existing address harvesting practices and cause similar collateral damage [SEL]. Message repositories and checksumming databases are brute force solutions with high false negative rates [MER]; in our experience they also require persistent high maintenance and ever-increasing resource utilization. The most recent attempts to add authentication into the mail delivery process through extending or replacing SMTP seem likely to be the most costly yet in terms of collateral damage and infrastructure costs [KNO], yet spammers are already observably capable of bypassing these measures using hijacked end-user machines to send messages using the local mail submission system and routing through authenticated channels, in the same way that email worms currently propagate [JdeBP]. This is not to say that these technical methods are entirely without merit or usefulness, as they are all effective in at least blocking some percentage of spam. However, the cost of these incremental cures is escalating to the point they may become as bad as the disease itself.

Legislative methods are in their infancy but are already being undermined by political pressures and seem likely primarily to force spammers to move even more of their operations to countries with friendlier laws, something they have demonstrated extreme willingness to do. Diligence on the part of prosecutors may yet produce results here, but they are likely to be a long time coming. The problem is here today, and its current severity can not be overstated. End users who are the worst affected are reporting hours spent per week deleting and attempting to block unwanted mail that is increasingly offensive in nature, and more and more are determining the effort of keeping their email usable is not worth it and are instead returning to other forms of communication.

When Paul Graham published his 2002 paper "A Plan for Spam" [GRA], which proposed applying Bayesian statistical modeling as a method of content filtering and provided very promising early results from Graham's own tests, many in the anti-spam and end-user communities lauded the approach as a possible permanent solution. Bayesian filtering claimed all the advantages of content filtering, while adding learning algorithms to defeat message character changes over

time and token-mangling attacks. It also promised an extremely minimal percentage of legitimate mail incorrectly classified as spam (false positives), and added a level of personalization in block lists that was unprecedented. Multiple Bayesian filter implementations appeared virtually overnight from both hobbyist and commercial sources, ranging from open source projects such as Bayespam and Bogofilter to implementations in Mozilla Thunderbird, Microsoft Outlook, and Apple's Mail program. Bayesian filtering fell from the limelight nearly as quickly, however, due to predicted difficulties in large-scale or centralized implementations, some success by spammers in defeating early Bayesian implementations using poisoning attacks, assumptions of vulnerabilities common among other content filters, and other issues both real and imagined. Academics and pundits continue to discuss the value and potentials of Bayesian methods, and Bayesian classifiers are still deployed alongside other filtering systems, but a popular opinion among administrators and the public seems to be that Bayesian filters are no better at providing a long-term solution to the problem than other methods [BAA, ALL, EMM, PAG, WAR, JdeBP2, BOW].

These dismissals are demonstrably premature, however. We implemented Bogofilter in a centralized capacity using common wordlists for an environment with thousands of accounts in April of 2003. Despite the best efforts of spammers to defeat it, Bogofilter has continually exceeded all expectations as a filter and has effectively solved the spam problem in our environment, allowing us to remove all other spam-specific filters from our architecture. Though we initially only set out to bring the problem under control and not necessarily to eliminate all spam from our environment, our estimates indicate we are currently able to accurately block 98-99% of the spam sent to us. The filter blocks an average of 1,100 incoming spam messages per hour (700,000 to 1 million spam messages per month), or 60-75% of our incoming mail volume. This amounts to roughly 40 spams per user per work day. We have vastly exceeded management's goals as well as our own, and our users are able to conduct business without constant solicitations appearing in their inboxes. Perhaps most importantly, only two business-related legitimate messages have been reported as blocked by this filter since it was implemented more than a year ago. While no solution is likely to last forever, our results indicate it is possible that properly configured and deployed Bayesian filters are capable of sustaining a high enough success rate that administrators may finally be able to stop spending all their time refining filters and instead focus on securing end-user machines, thus finally moving to take the offensive in the spam war.

Environment and Goals

Company and Environment

Our company, VHA Inc., is an alliance of not-for- profit hospitals, health systems and their affiliates. Member organizations range from single 50-bed facilities to large, integrated health care systems made up of multiple hospitals, physician clinics, and support care sites. Notable member organizations include Baylor Health Care System, Cedar-Sinai Health System, Mayo Foundation, and Yale-New Haven Health Services Corp.

The mail environment in question provides mail services for more than 2,000 accounts distributed between our corporate headquarters and approximately 30 regional offices nationwide. Monthly mail volume over the past year averaged 1.85 million messages per month, 70% incoming and 30% outgoing. Based on filter logs and user feedback, we estimate today that on average 65-70% of the incoming mail (nearly 50% of the total mail volume) is spam. The content of our mail is typical of any company our size; the one exception is that since we operate in the health care industry and specifically the medical supply purchasing industry, we see quite a lot of legitimate mail having to do with pharmaceutical contracts and supplies, i.e., messages mentioning Viagra and other drugs can and do show up in our legitimate mail flow.

Architecture

All mail entering or leaving the environment passes through a centralized pair of load-balanced mail exchangers running the qmail MTA. The first exchanger currently has dual 1 GHz Pentium III's, the second has dual 700 MHz PIII's. Both have 1 GB RAM. Incoming mail is handed off to a Microsoft Exchange system which also handles all internal mail (Figure 1). Outgoing mail originating at the Exchange system is handed to the mail exchangers directly, while mail originating from internal applications uses a separate internal qmail server for relaying to the outside. All of our spam filtering efforts have been targeted at the qmail environment due to resource utilization issues and end-user interaction concerns.

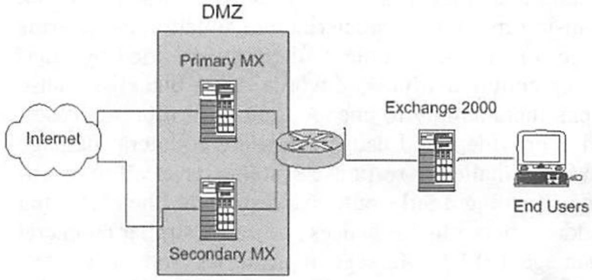


Figure 1: Mail environment architecture before Bogofilter.

The current Bogofilter implementation added one additional server to our environment. This server is responsible for caching mail as required for filtering purposes, receiving end-user filtering corrections, providing the environment that administrators use for processing corrections and ongoing training of Bogofilter, and holding the master copy of the wordlists (Figure 2). This server currently has dual 2 GHz Xeons and 2

GB RAM, but rarely has a CPU load average higher than 0.1. More information about this server is provided in the "Design" section below.

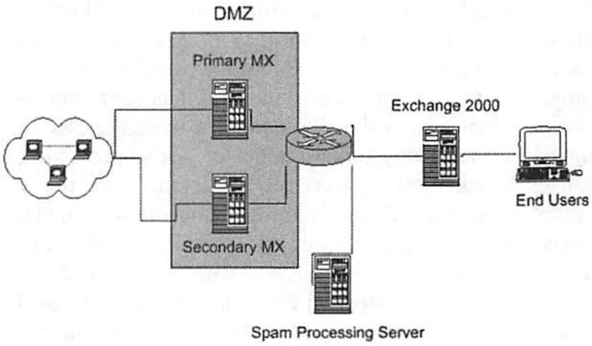


Figure 2: Mail environment architecture after Bogofilter.

Goals

Our goal was not to eliminate spam completely from our environment, but to bring the problem under control so that normal work could go on unaffected. The initial target was to block 75% of our spam, which we initially estimated would be 30% of our incoming mail volume. We also needed to guarantee that we would not block any legitimate business mail in the process of blocking spam. Given our mail volume, anything we implemented needed to be fast and efficient enough not to make excessive demands on our existing infrastructure. As a final goal, we wanted to keep the blocking centrally managed at the server level, rather than something that the users would have to deal with.

Initial Filtering Attempts

Basic Filters

Basic initial attempts at blocking via fixed-string matches against the sender address, reverse DNS lookups, and even a short-lived block of mail from all non-US domains met with predictable results. We quickly learned that the spammers we were dealing with were not just sending indiscriminately to global email lists, but were specifically monitoring the delivery status of spam entering our environment and were willing and able to change their messages to avoid our filters. We were not willing to spend hours each day creating new filters which would be made obsolete within a week, so we began to look in earnest for a more viable long-term solution.

Due to the nature of the spam arms race and our goal of efficiency we decided to target our efforts at automated content-based filtering. Blacklists were not considered an option due to the requirement to avoid blocking any form of legitimate mail. Challenge-response systems placed unacceptable burdens on our business contacts while simultaneously being considered too easy to spoof. More dramatic efforts aimed at user and mail route authentication seemed unlikely to provide any long-term relief, since we assumed they would primarily force spammers to continue their recent attacks on user computers themselves, using zombies and the local user's own credentials to send spam through valid mail routes. Spam is not spam without the content, however, and content-based filtering appeared to offer the most desirable combination of accuracy and tunability. Scalability was the most likely point of failure, but we hoped that a sufficiently automated system could handle the load.

Vipul's Razor

Vipul's Razor, commercially distributed through Cloudmark, was selected for a pilot. This is a checksumming content filter; each incoming mail is compared over the network in real time to a database of known spam messages. This database is fed by user submissions and spamtrap addresses maintained by Cloudmark. The advantages of this type of system are that it has a negligible false positive rate and is relatively difficult to attack directly. The primary working attack is to find a way to pollute the spam database or otherwise disrupt the checksum sharing process. The disadvantages include scalability and ongoing maintenance; someone, somewhere, must receive each new spam before it can be stored for future comparison.

While the implementation was effective in blocking 30% of our incoming mail volume as spam, it introduced significant system overhead and proved ultimately unscalable. This was primarily due to the extra network traffic required to contact the checksumming database and, even more significantly, the Perl instantiation required per message on the mail exchangers for the Razor client. Our environment experienced more service delays and outages than it ever had previously, due to both intermittent network errors in reaching the database servers and excessive CPU load produced by the clients.

Most important, however, was the fact that our end users reported no noticeable change in the volume of spam they were dealing with, despite the 30% reduction in overall mail volume. Complaints of offensive spam during this period actually increased. This was our first indication that our initial estimate of the size of our problem was off by a wide margin. We considered moving to a local checksum-based solution such as DCC, but eventually concluded that this method was too high maintenance to be viable in our environment. We began researching other alternatives, with even more attention paid to the real world system requirements imposed by prospective solutions.

Bogofilter Implementation

Bayesian Classification and Filtering

Although there was work being done on Bayesian classification of spam as early as 1996, Paul Graham's paper [GRA] is generally credited as being the first description of a solid implementation with

promising results. Gary Robinson later improved on Graham's work, suggesting algorithm modifications to make Graham's technique mathematically consistent with Bayesian statistics [ROB]. Robinson's probability-combination improvements make use of the inverse chi-square function first described by Ronald Fisher and are therefore sometimes referred to as the Robinson-Fisher algorithm.

Probability theory holds that the probability of a given event can be plausibly estimated by observing how often it has occurred in the past under similar circumstances. Bayesian probability begins by creating a "prior probability distribution," or "prior," which estimates the probability of given events. The prior can be used to calculate the likely outcome of new events. Then experiments are run and the outcomes recorded. The prior can then be updated to reflect the outcome of the experiments. Bayesian algorithms are self-correcting in this respect: they learn from their mistakes.

Token Scores

If an email is viewed as a collection of discrete words, or "tokens," a score can be assigned to each token. These scores are roughly comparable to probabilities in that they directly correspond to a token's "spamminess" or "non-spamminess." Each token's score represents how likely that token is to appear in an email composed of tokens that are uniformly distributed and statistically independent. The more "spammy" or "not spammy" a token is, the less likely it would be to appear in such an archetypal email. That is, the presence of these "interesting" tokens in an email violates statistically neutral linguistic behavior in measurable ways.

These scores are easy to calculate given a relatively even number of manually sorted spam and non-spam emails; the math is described in detail by Robinson in his paper "Spam Detection" [ROB]. The tokens themselves, plus the frequency with which they occurred in the text of spam and non-spam messages previously used to train the filter, are used to create a prior probability distribution in database form. Most Bayesian spam implementations are "objective" in that they put a lot of thought into assigning the prior, especially for tokens that do not currently exist as part of the prior. Bogofilter is no exception. When it encounters a new token, it assigns it the value of the variable `robx` ("Robinson's x"). `robx` is calculated as the average of the scores of all the other known tokens. The variable `robs` ("Robinson's s") is used in situations where Bogofilter has only seen the token a few times (low data situations). `robs` acts as a user-defined metric of trust and limits `robx`'s effect in low data situations.

Combined Scores

Once the prior is used to calculate a score for each word in an email, these token scores must be combined into a single score, which is representative of whether or not the given email is spam. Bogofilter uses the Robinson-Fisher algorithm for probability combination by default. There are three important aspects to the inverse chi-square driven algorithm Robinson has provided. First, it removes assumptions that were not relevant in the context of spam filtering, such as assuming the probability of a given token's prediction being correct is the same whether its outcome is spam or non-spam. Second, it is more sensitive to token scores that indicate if a message has an underlying tendency toward spam or not spam. Third, it uses a user-defined variable to decide how many "interesting tokens" to combine, which more consistently handles both large and small emails. For Bogofilter this variable is named `min_dev`. `min_dev` is the minimum deviation from the neutral score of 0.5 a token must have to be considered "interesting" and therefore be included in the classification of the message as a whole.

The other user-defined variables exist in the form of cutoff values to which the combined score of the email is compared. In binary classification, emails with scores over a singular threshold are considered to be spam. Those under it are not. Bogofilter optionally allows for three-factor classification. Two cutoffs are provided, `spam_cutoff` (for spam) and `ham_cutoff` (for non-spam). Anything in between is labeled "unsure." In practice, "unsure" mails provide interesting fodder for training, so this is the classification method we use.

Selection of Bogofilter

Graham's paper had been published for some time at this point, and like the rest of the industry, we were intrigued by his reported success. The concept of a Bayesian approach appealed to us, given our experience with shifting spam patterns and our desire to stick with content-based filters. The nature of the filtering, however, predicted the best results when each user maintained individual wordlists to most accurately reflect the unique nature of individual mail spools.[1] Our goals of scalability and minimal user intervention were at odds with this, but we decided to do some initial testing to see if it could work with a single set of wordlists for an entire organization.

We began monitoring the field of filters claiming a Bayesian implementation, and quickly settled on Bogofilter. Although other implementations had features Bogofilter at the time lacked, it was an obvious choice to us for its small overhead and its attempt to work within the Unix philosophy of "do one thing and do it well." Bogofilter is written in C and expects to operate on standard I/O streams, adding a custom header to messages it operates on and/or indicating message status with an exit code. This fit our environment perfectly. There are other Bayesian filtering

[1] Ironically, a primary impetus for the original creation of Bogofilter was to create a filter which implemented Graham's proposals but could be quickly deployed by individuals [ESR].

systems which tend to worry less about efficiency than dealing with cutting edge theory; these are ideal for experimental approaches and furthering the state of the art but are less useful in a high-volume production environment. This is not to say that Bogofilter's implementation is not correct, however; the Bogofilter development group expends a good deal of effort ensuring that their implementation of Bayesian algorithms is correct, as well as tracking changes and advances in Bayesian filtering theory and providing their own measurements and contributions to the field. Bogofilter strives to provide the best of both efficiency and accuracy and was therefore a good choice for our environment. Selection of this tool has no doubt contributed heavily to our filtering success.

Training

Another likely source of our success is our training process, both for initial wordlist seeding and subsequent training and corrections. Bayesian filtering is unfortunately not a turnkey-style solution; while it is possible to implement a Bayesian spam filter (including Bogofilter) by simply following the steps in a HOWTO and running some scripts, best results require that the administrators have some understanding of the theory and how best to apply it to their environment. This is primarily relevant in the initial and ongoing training process. We spent a fair amount of time gaining an understanding of the theory and the work being done to apply it to spam filtering and Bogofilter's specific implementation before moving to implement, and we designed our training process accordingly.

Sorting

Proper training requires a large pre-sorted collection of spam and non-spam messages. It is nearly impossible to create an effective Bayesian classifier using only a handful of mails. The filter needs to be trained on at least several thousand messages each of spam and non-spam from the start, preferably in nearly equal amounts [LOU].

To seed the initial wordlists we therefore collected several days' worth of incoming and outgoing mail at the mail exchangers. This provided us with more than 220,000 messages, which we then classified manually into groups of spam and non-spam. Approximately half of these were discarded as mailer daemon traffic such as bounces, delivery overhead, and virus quarantine messages. All outgoing mail was automatically classified as non-spam but was kept to provide a large body of known good mail so that if the filter established any biasing error it would be in favor of keeping legitimate mail. The remaining mail was classified in successive phases. Mails were manually sorted by one administrator while another looked for patterns in the already classified mail to allow for batch classifications to speed up the process. These batch filters were similar to the ones employed by other anti-spam software; while not long-term options for our environment, they worked in the short term due to the static nature of the mail snapshot we were operating against and the lower efficiency requirements. Anything faster than a human was a benefit here. Further, this processing was done in a development environment away from the regular mail exchangers, so regular mail load was not affected. We also took some time to develop interactive shell scripts to aid the manual classification process, both to speed it up and to preserve as much privacy as possible. The scripts provided only the headers of messages, color-coded to highlight suspicious patterns. The full messages were available at operator request, but were rarely needed in the initial classification stages.

After approximately 30,000 messages had been sorted, we began to incorporate Bogofilter itself as a sorting tool. We tested its effectiveness by running it against 20,000 pre-sorted messages, 10,000 each randomly taken from the sorted collections of spam and non-spam. Bogofilter without any established wordlists was presented with random individual messages from these collections and asked to determine if they were spam or non-spam. If its classification agreed with the human classification, no action was taken. If its classification disagreed with the human classification or Bogofilter was unsure of how to classify the message, it was corrected based on the human classification (this process is known as "train on error"). The output of the classification versus the human classification was presented in real time during the test run to an administrator (Figure 3). At the end of the run, messages Bogofilter had consistently gotten wrong were further investigated.

Results at this early stage were simply shocking. At the start of the run Bogofilter would tend to get a few messages wrong until it had a handful of each type of message in its wordlists, at which point it would immediately begin to get the vast majority of classifications correct, increasing dramatically and observably until it had seen several thousand messages, at which point it would generally stabilize at around 95% accuracy. Inaccuracies were either false negatives or "unsures" in all but a handful of cases. More often than not messages which Bogofilter persistently had trouble classifying were re-evaluated to find that the human administrator had gotten them wrong in the first place and Bogofilter was correcting us.

Once we had performed several of these test runs we were confident Bogofilter could be utilized as a sorting tool. We created wordlists based on all the messages sorted so far and then ran Bogofilter against the remaining unsorted lists, allowing it to sort them into provisional groups. These groups were then further evaluated manually by an administrator to verify the accuracy of their sorting before they were added to the global collections.

Eventually, nearly 20,000 messages were removed due to excessive ambiguity about their nature.

These included news updates, potentially legitimate commercial mailings, religious and inspirational messages, joke-of-the-day lists, and others. We decided it would be better to move forward with these messages unclassified and allow users to give more feedback during the pilot phase of the process.

This sorting process took both administrators the better part of a week to complete, and we became intimately familiar with the character of the spam we were receiving. This was very tedious work, but there is no doubt taking the time to do this was a key to the current success of our implementation.

Configuration and Tuning

Once the initial message sorting is complete, the filter must be configured and tuned for its environment. There are several variables to consider, along with their interaction, and we attempted to use appropriate values for each during each stage of the training process. On some occasions we did test runs using various combinations of settings to see which provided the best results. The theories involved were still being formed and actively debated, so we were experimenting pragmatically to find the best settings, but in most cases the theoretical and empirical work that has been published since agrees with our results. On a related note, even though Bogofilter at the time shipped with tools to perform all of this training and tuning, these proved too nascent and incomplete to meet our needs, so we developed our own. Current versions of Bogofilter ship with complete training tools which provide even more rigorous tuning functionality than what is described here. We are also deeply indebted to Greg Louis for his excellent tuning documentation [LOU].

```
% head -n 25 t.log | sed -f t.sed
   NOTSPAM/1051176904.31414.XXXXXXXX: NOTSPAM:  Legit:  spamicity=0.00e+00
      SPAM/1051295846.6703.XXXXXXXX:     SPAM:   Spam:  spamicity=1.00e-00
   NOTSPAM/1051277931.15769.XXXXXXXX: NOTSPAM:  Legit:  spamicity=0.00e+00
   NOTSPAM/1051198384.24223.XXXXXXXX: NOTSPAM:  Legit:  spamicity=7.37e-04
      SPAM/1051151402.7668.XXXXXXXX:     SPAM:  Legit:  spamicity=8.77e-03
   NOTSPAM/1051316490.31430.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.476275
   NOTSPAM/1051212317.20791.XXXXXXXX: NOTSPAM:  Legit:  spamicity=7.00e-06
   NOTSPAM/1051198083.18275.XXXXXXXX: NOTSPAM:  Legit:  spamicity=1.66e-04
      SPAM/1051226341.24583.XXXXXXXX:    SPAM: Unsure:  spamicity=0.499488
   NOTSPAM/1051160935.17463.XXXXXXXX: NOTSPAM:  Legit:  spamicity=2.62e-03
   NOTSPAM/1051300472.26807.XXXXXXXX: NOTSPAM:  Legit:  spamicity=3.63e-03
      SPAM/1051307951.28054.XXXXXXXX:    SPAM: Unsure:  spamicity=0.500000
      SPAM/1051268971.18425.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
   NOTSPAM/1051223545.17452.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.500000
      SPAM/1051187116.12926.XXXXXXXX:    SPAM: Unsure:  spamicity=0.500210
      SPAM/1051176998.2277.XXXXXXXX:     SPAM: Unsure:  spamicity=0.499942
   NOTSPAM/1051228484.31637.XXXXXXXX: NOTSPAM:  Legit:  spamicity=2.13e-14
   NOTSPAM/1051185896.25758.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.455169
      SPAM/1051246738.10352.XXXXXXXX:    SPAM: Unsure:  spamicity=0.500008
      SPAM/1051280016.3280.XXXXXXXX:     SPAM: Unsure:  spamicity=0.503970
   NOTSPAM/1051199776.12275.XXXXXXXX: NOTSPAM:  Legit:  spamicity=5.12e-05
      SPAM/1051284997.14278.XXXXXXXX:    SPAM: Unsure:  spamicity=0.572632
      SPAM/1051302521.23943.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
   NOTSPAM/1051284875.12731.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.303452
   NOTSPAM/1051220593.13157.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.394604
%
% tail -n 25 t.log | sed -f t.sed
   NOTSPAM/1051210780.27932.XXXXXXXX: NOTSPAM: Unsure:  spamicity=0.129265
      SPAM/1051175772.21302.XXXXXXXX:    SPAM: Unsure:  spamicity=0.663503
      SPAM/1051257860.3202.XXXXXXXX:     SPAM:   Spam:  spamicity=9.99e-01
   NOTSPAM/1051193288.23220.XXXXXXXX: NOTSPAM:  Legit:  spamicity=1.06e-10
   NOTSPAM/1051193593.21449.XXXXXXXX: NOTSPAM:  Legit:  spamicity=0.00e+00
      SPAM/1051136802.3620.XXXXXXXX:     SPAM:   Spam:  spamicity=9.98e-01
   NOTSPAM/1051208078.15694.XXXXXXXX: NOTSPAM:  Legit:  spamicity=7.11e-03
   NOTSPAM/1051138154.23411.XXXXXXXX: NOTSPAM:  Legit:  spamicity=0.00e+00
      SPAM/1051175724.22780.XXXXXXXX:    SPAM:   Spam:  spamicity=9.59e-01
    NOTSPAM/1051276954.4253.XXXXXXXX: NOTSPAM:  Legit:  spamicity=9.08e-12
      SPAM/1051266551.15139.XXXXXXXX:    SPAM:   Spam:  spamicity=9.75e-01
     NOTSPAM/1051259750.482.XXXXXXXX: NOTSPAM:  Legit:  spamicity=8.68e-04
      SPAM/1051252160.15717.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
      SPAM/1051225816.5567.XXXXXXXX:     SPAM:   Spam:  spamicity=9.84e-01
      SPAM/1051179746.27924.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
      SPAM/1051268772.31580.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
   NOTSPAM/1051301801.15206.XXXXXXXX: NOTSPAM:  Legit:  spamicity=2.79e-09
   NOTSPAM/1051191245.27903.XXXXXXXX: NOTSPAM:  Legit:  spamicity=1.42e-13
      SPAM/1051205684.30817.XXXXXXXX:    SPAM:   Spam:  spamicity=1.00e-00
      SPAM/1051287773.16831.XXXXXXXX:    SPAM:   Spam:  spamicity=9.91e-01
      SPAM/1051316745.1425.XXXXXXXX:     SPAM:   Spam:  spamicity=9.69e-01
    NOTSPAM/1051203657.6582.XXXXXXXX: NOTSPAM:  Legit:  spamicity=8.39e-09
      SPAM/1051233957.9233.XXXXXXXX:     SPAM: Unsure:  spamicity=0.830934
      SPAM/1051291673.32736.XXXXXXXX:    SPAM:   Spam:  spamicity=9.59e-01
      SPAM/1051251862.11339.XXXXXXXX:    SPAM:   Spam:  spamicity=9.99e-01
%
```

Figure 3: Screenshot of the beginning and end of a preliminary Bogofilter classification run of our sample mail corpora.

As noted above, Bogofilter's primary configuration variables in tri-state classification mode are `robs`, `robx`, `min_dev`, `spam_cutoff`, and `ham_cutoff`. `spam_cutoff` and `ham_cutoff` default to 0.95 and 0.10, respectively. For the sorting process above we reduced `ham_cutoff` to 0.05 to require a higher level of certainty from Bogofilter before it was allowed to flag a message for the legitimate corpus. For the training process we also used 0.05 to inflate the number of legitimate mails that would be classified as "unsure" and therefore used for training during the "train on error" process. These were more safeguards to ensure any biases introduced during training would be on the side of blocking too little mail instead of too much.

To determine appropriate values for `spam_cutoff` and `robs`, we ran several iterations of a training script using a methodology similar to recommendations published by Greg Louis [LOU2, LOU3]. We divided our pre-sorted message collections into three randomized groupings. Group A had 20,000 messages (10,000 each of spam and non-spam). Group B had 10,000 messages (5,000 each of spam and non-spam). Group C had all remaining messages (approximately 45,000). Bogofilter was fully trained on each message in Group A, then trained on error for each message in Group C. Group B was then used as a test corpus, with no training done and errors tabulated. The train-on-error and test runs were repeated once. Following this, Group B was used to further train on error, then again used to test; this process was also repeated once. In this fashion Bogofilter was either fully trained or twice trained on error for every message in our corpora. See Appendix A for more detail.

We needed to determine if we should use the default `spam_cutoff` of 0.95 or if we could safely use a more aggressive value of 0.90. We also needed to determine if we should use a `robs` value of 0.01 or a more conservative 0.001.[2] We therefore ran the above test suite for each of the four permutations of these two values. While each of the four gave nearly identical results by the final two test rounds, the 0.01 and 0.90 combination had the highest accuracy during the initial rounds and was therefore selected (Figure 4). It was a pleasant surprise that using a lower `spam_cutoff` actually improved accuracy; apparently a statistically significant amount of our spam scored between 0.90 and 0.95, and using the lower cutoff meant more of these were classified correctly on the first pass.

For `robx`, we again cleared our wordlists and ran several iterations of a similar training script, beginning with fully training on 10,000 each of spam and non-spam, then training on error for the rest of the messages. At the end of each iteration we took the final `robx` value and used it as the initial `robx` value for the next iteration. We did this several times, until the `robx` value stabilized at 0.477112.

The final value is `min_dev`. We left this at the default of 0.1 throughout training and into production. The Bogofilter tuning documentation recommends moving this to somewhere between 0.3 and 0.46 to take into account fewer words per message, but testing

[2] At the time there was some debate about whether overly conservative `robs` values might cause unpredictable results. Louis has since conclusively determined that values lower than 0.01 do cause problems if a token occurs a few times in one wordlist but not at all in the other [LOU4].

robs/spam_cutoff	round	fpos	fneg	uspam	unotspam	err	percent
0.001/0.95	1	3	5	104	64	176	1.76%
	2	3	5	107	63	178	1.78%
	3	0	0	14	10	24	0.24%
	4	0	0	10	5	15	0.15%
0.010/0.95	1	3	7	87	47	144	1.44%
	2	3	8	83	48	142	1.42%
	3	0	0	15	11	26	0.26%
	4	0	0	10	4	14	0.14%
0.001/0.90	1	3	5	88	65	161	1.61%
	2	3	5	90	63	161	1.61%
	3	0	0	11	11	22	0.22%
	4	0	0	10	6	16	0.16%
0.010/0.90	1	3	8	72	46	129	1.29%
	2	3	8	70	47	128	1.28%
	3	0	0	11	11	22	0.22%
	4	0	0	10	5	15	0.15%

```
fpos    : false positives
fneg    : false negatives
uspam   : unsure (spam)
unotspam: unsure (not spam)
err     : total errors
percent : percent error
```

Figure 4: Results of varying the value of `robs`and `spam_cutoff`.

on our part showed that this reduced accuracy, and recent experiments with raising the value to 0.3 resulted in an immediate and dramatic increase in the amount of spam that made it through the filter. However, we will likely need to revisit this as wordlist attacks become more focused.

At this point the tuning was considered complete. Apart from raising the `ham_cutoff` back to 0.10 (to avoid an excessive number of "unsure" classifications), these were the wordlists and configuration we took to production (there are other options such as how the lexer should tokenize HTML tags and IP information; we have left these at the defaults). Throughout the tuning process we remained amazed at the speed with which Bogofilter gained accuracy during the training runs, and concerns that using a single set of wordlists for an organization of this size would lead to significant misclassifications appeared unfounded. Although we tried to keep in mind that our results were preliminary, saying that we had significant remaining doubts by the end of the training iterations would be a mischaracterization.

Design

As noted above, our border mail architecture consists of a pair of mail exchangers running qmail. Prior filtering attempts ran as mirrored installs on both of these servers. Bogofilter, however, required some method of maintaining the same word list on both servers and any future servers added for expansion purposes. We also needed a central location to receive end-user misclassification notices so that an administrator could process them and train Bogofilter on filtering errors. We knew this central host would likely need to be able to perform extensive training and other processing operations which we would not want to impact our general mail load. None of this processing, however, needed to happen in real time for the mail to be properly filtered. We therefore added a new server to provide for general spam processing. When filter corrections are necessary, the end user forwards the misclassified message to this server, where an administrator verifies the request and trains Bogofilter, updating the central copy of the wordlists which are stored on this server. These wordlists are updated on the mail exchangers nightly. Since the mail exchangers have their own copies of the wordlists, there is no effect to mail flow if this server is down or otherwise unreachable. Finally, the spam server maintains data on all the training operations that have occurred so that the wordlists can be recreated from scratch as required.

The actual filtering happens at the mail exchangers during the SMTP exchange. We use the qmail-qfilter wrapper around qmail-queue to provide real-time message filtering options. Our qfilter invocation first notes the mail in the filtering logs, then pipes the message through Bogofilter, then copies the mail to the spam server cache, then checks the Bogofilter-generated spamicity header to determine whether or not the message is spam (Bogofilter is also able to indicate message status with an exit code, but passthrough mode and header-based filtering proved more compatible with the qfilter and spam caching server implementation). If the message is spam, qfilter exits 31, which causes qmail to refuse to accept the message from the originating server. See Appendix B.

The solution of refusing to accept spam mail during the SMTP exchange has become somewhat popular because it dodges the problems of network congestion and server load created by attempting to send bounce messages to senders that do not exist. We also chose it because in the case of false positives it lets the legitimate sender know immediately that their message was refused, allowing them to follow up quickly with their intended recipient. External relay servers may in theory still create unnecessary bounces or cause delays in receipt of legitimate bounces, but this was determined the best fit for our environment. However, if spammers begin to train "evil" Bayesian filters to attack "good" Bayesian filters (as John Graham-Cumming has suggested [JGC]), we will need to reconsider any implementation aspects such as this one which indicate to spammers the real delivery status of their messages.

To create our ongoing training framework we augmented the scripts we had written for the initial wordlist creation. Mails users submit for correction are viewed on the spam server by the administrators either interactively in Mutt or using a custom script that iterates across the entire queue. In either case, the administrator is presented with the mail headers and information on how Bogofilter classified the message initially and how it is currently classified (Figure 5). In some cases Bogofilter will have changed the way it classifies a given mail based on prior corrections, and the message can be skipped. If Bogofilter still misclassifies the message, the administrator can view the message headers and body and provide correction as required. The training has been deliberately kept as a manual process to prevent user error from corrupting the wordlists. We also do not use Bogofilter's auto-update (-u) switch, as this is likely to introduce a significant amount of error in a high-volume environment.

While testing this configuration, the primary issue that had to be resolved related to the fact that the users were forwarding messages for correction after Exchange had delivered them. Exchange modifies the message headers (and in some cases the body) significantly, which means that the messages users forward us are not the same messages Bogofilter originally classified and are useless for training. The best solution to this seemed to be to cache the mails as Bogofilter originally saw them and look them up as required. The mail exchangers therefore forward a copy of all incoming messages to the spam server, where they are cached for a period of two weeks. The training lookup

scripts take the mails users forward, extract several pieces of header data, attempt to reconstruct the original headers, and look up the original message in the cache using these headers. Message headers are cached parallel to the full bodies to speed up lookups. This is quite frankly the least elegant piece of this entire filtering system, but no other option has been discovered for easily dealing with Exchange's header-mangling problem.

See also Greg Louis' article "Is Bogofilter Scalable?" [LOU5], which was written at approximately the same time as we were developing this implementation, and discusses many of the same issues. While Louis ends with uncertainty that the implementation he describes would scale enough to be able to handle an environment 10 or 50 times the one he describes (3,500 or 17,500 spam messages per day, respectively), our implementation demonstrates that a similar setup can scale to at least 75 times the spam volume he describes. We are not verifying all of our classifications as rigorously as he suggests, but so far this has not created a problem.

Implementation

The Bogofilter roll out to production was to occur in four phases. Phase 0 was a two-week trial by the CIO for his mail only, primarily so that he could verify no mail would actually be blocked until further testing had been done. Phase 1 was a one-month user-interactive pilot with the entire MIS department (approximately 80 users) participating. Phase 2 was a three-month user-interactive period with the entire company asked to participate. Phase 3 implemented Bogofilter in a decision-making capacity such that it began actively preventing the delivery of messages it classified as spam.

For Phases 0, 1, and 2, Bogofilter was implemented in passthrough mode alongside the existing Vipul's Razor deployment. Any mails that Vipul's did not block were filtered through Bogofilter, and a header was added with a classification for the "spamicity" of the message in numeric form. Outlook rules were created and distributed to the pilot groups to sort mails classified as spam and unsure into their own folders. Users were asked to review the contents of those folders on a periodic basis. Any legitimate mails found in the spam or unsure folders were to be forwarded to one address, and any spam messages found in the inbox or unsure folders were to be forwarded to another address. Administrators monitored those addresses and corrected Bogofilter on mails that had been classified incorrectly as described in the "Design" section above.

Some small training complications resulted from our attempt to maintain a single, company-wide set of

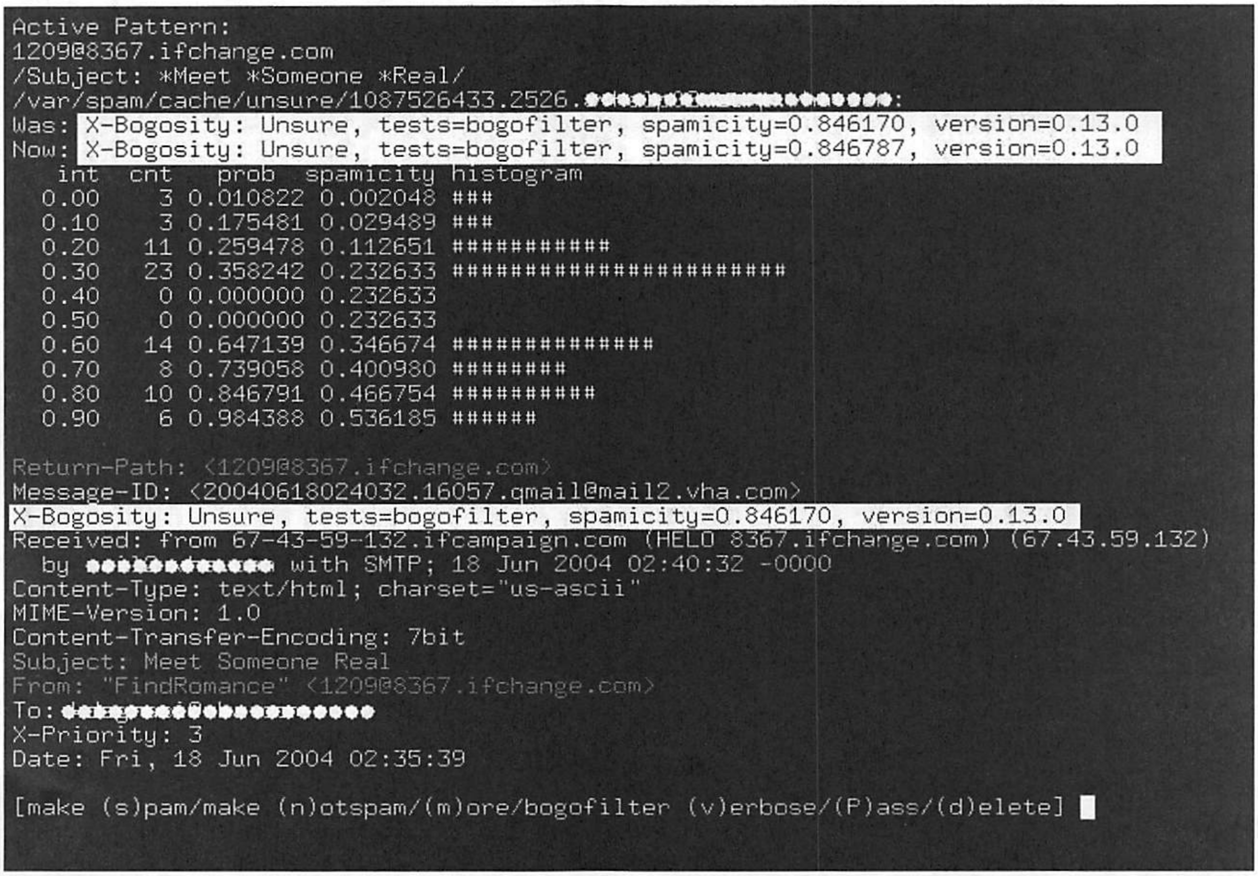


Figure 5: Screenshot of a spam message being displayed in our training tool.

wordlists. In reality, there is a large grey area between "spam" and "non-spam" with a user population of this size. "Joke of the day" lists, airline/hotel advertising, and religious messages are some examples. Worse, end-user collisions were initially frequent, where two users found the same piece of bulk mail in their unsure folders and one would report it as spam while the other reported it as non-spam. In an extreme case one user reported mail from the same home-improvement list as either spam or not spam, based on the home-improvement advice in question (roofing advice was spam, gardening advice was not). These were generally resolved by the administrators on a case-by-case basis, usually by either letting mob-rule prevail or allowing Bogofilter to decide based on the content of the mail in question. In the case of the home-improvement list, the administrators did as the user instructed, and Bogofilter adjusted accordingly.

Despite these minor complications, the pilot phases were an unqualified success. All but a handful of users reported all the spam gone from their inbox into either the spam or unsure folders. Though an implementation goal was to avoid user interaction and the final production system primarily does that, users seemed to appreciate the brief opportunity to do something about the spam that was reaching their inboxes. Most importantly, users were able to observe the filter in action and confirm first-hand that legitimate mail was not being misclassified; even during the pilot phases, no legitimate business-related mail was reported as misclassified by the filter. Finally, the administrators were confident at the conclusion of the pilot period that the maintenance and ongoing training framework was scalable and practical for the task at hand.

Once all parties were satisfied with the accuracy of the solution, the mail exchangers were reconfigured to use Bogofilter's spam classifications to block mail.

Results and Observations

Since its deployment in a blocking capacity, Bogofilter has continued to provide unprecedented accuracy and efficiency in mail filtering. Since the beginning of our spam mitigation effort, incoming email volume has risen from 900,000 to 1.4 million messages per month. Bogofilter has consistently

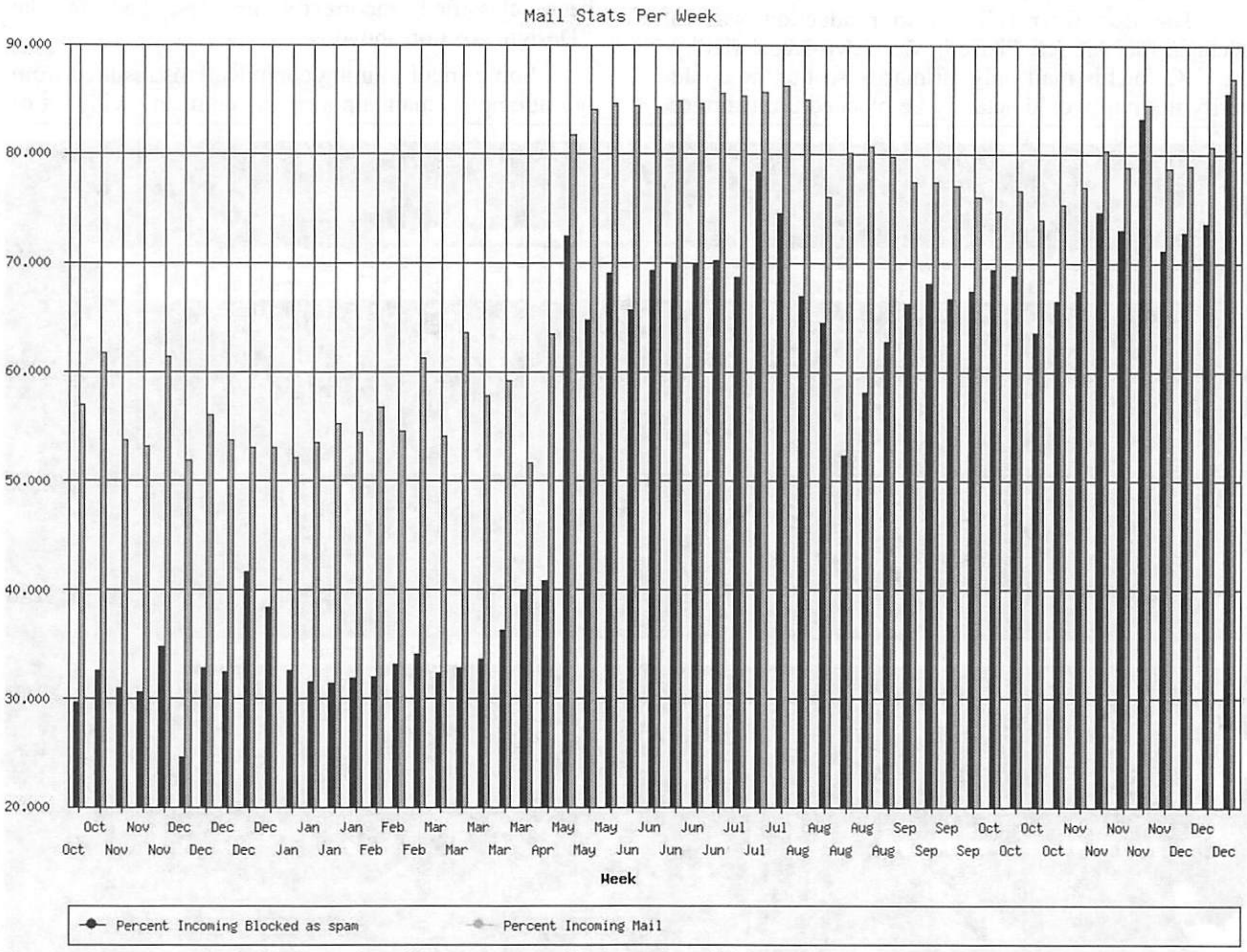


Figure 6: Percentage of inbound mail/Percentage of inbound mail blocked as spam, October 2002-October 2003. Bogofilter was installed in late April 2003. As we refused delivery of spam, we bounced less mail. As we bounced less mail, the outbound percentage dropped, and inbound percentages increased accordingly.

blocked 60-75% of this incoming mail as spam, or between 700,000 and 1 million messages on average per month. The filter averages 1,100 blocked mails per hour at the exchangers; when the average number of recipients per blocked mail is factored in, this is roughly 40 spams per user per work day. While it has been difficult to quantify exactly how much spam still enters our environment, based on end-user reports, mail logs, and manual inspection we believe the filter is 98-99% effective. This performance has persisted without fail for more than a year (Figures 6 and 7).

Further, in that time there have been only five blocks reported as false positives, only two of which were legitimate filter misclassifications and business related. One of these false positives involved a user mistakenly reporting legitimate mail as spam, then having similar followup messages blocked as spam. Another involved a group of users reporting a group of related mails as spam, then having a new user attempt to receive mail from that same source and having it blocked. A third was a user who had personal mail regarding a PayPal transaction blocked; as personal PayPal mails are not a common occurrence at our company, most of Bogofilter's opinion of PayPal tokens had been derived from spam categorization of PayPal phishing scam mails. All of these were resolved through either correcting the previous misclassifications or training on the blocked mail.

The two remaining misclassified messages were real false positives. The first was sent from a vendor providing an employee reward program. Though this was a legitimate business mail, its content was virtually indistinguishable from spam (e.g., "someone has sent you $xx.xx, click here to redeem it"). The other false positive was a bulk employee satisfaction survey sent from an external source. Some recipients received this mail fine, while others had it blocked. Inspection revealed that for some reason some of the mails had been sent with both Spanish and English language parts, while the rest were English only. The English mails got through, while the Spanish mails were blocked because those tokens had previously been encountered primarily in spam messages.

Since going to production we have not experienced any of the predicted downsides of using a single set of wordlists with centralized administration. It is possible that our user population's legitimate mail is more homogeneous than that of an average organization, but this seems unlikely. Rather, expectations that the filter's accuracy would "fall apart" when the same wordlists were used for many users seem simply not to have been borne out in practice. While even a success rate of 99% is an order of magnitude lower than the 99.9% and higher rates individual users of Bayesian filters have achieved, this does not necessarily indicate an inherent loss of discrimination ability due to shared

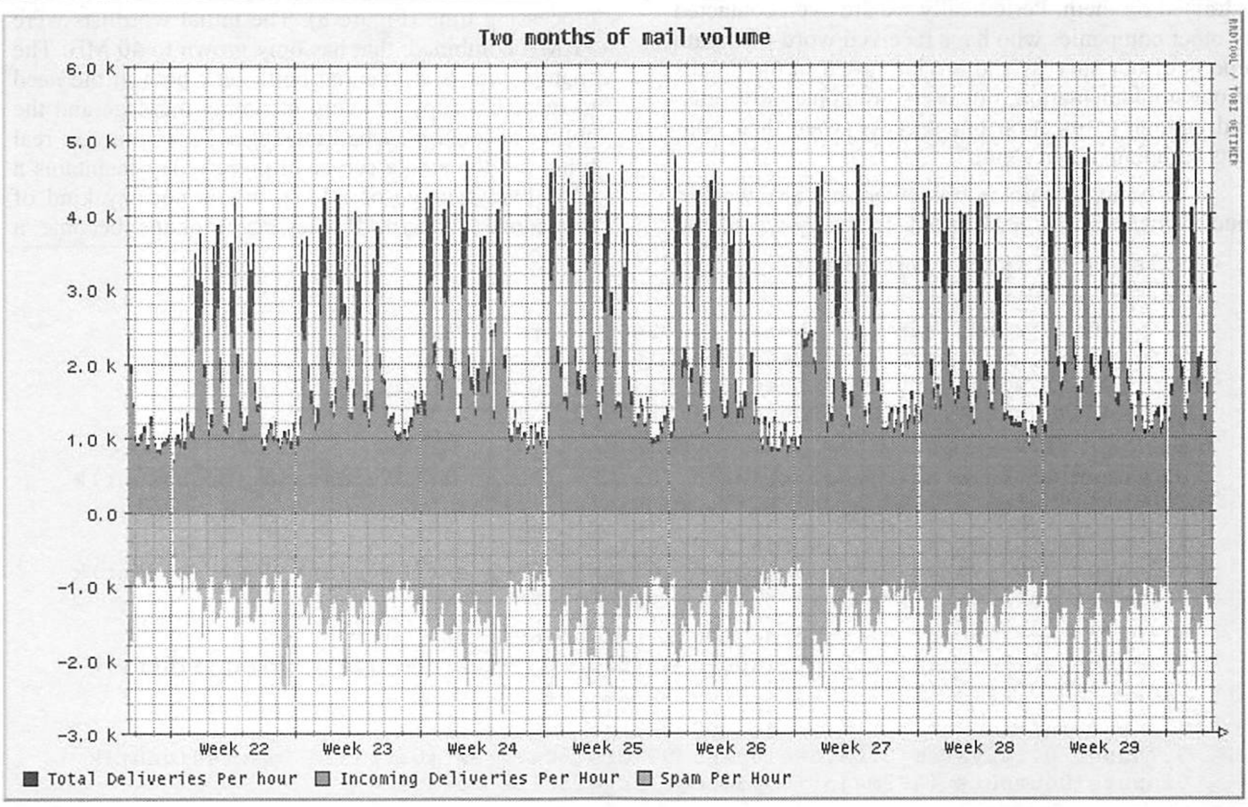


Figure 7: Overall mail traffic, August 2003-April 2004. Spam is multiplied by negative 1, to make the graph more readable. Data gaps are caused by log rotations.

wordlists. It could instead be the result of only training on the filtering errors users report instead of on all errors. In any case, blocking 99% of our spam is more than sufficient to meet the needs of an organization such as ours, and much better than any other solution we have evaluated or considered. The other minor wordlist-sharing issues encountered during the pilot of some users reporting mail as spam and other users reporting the same mail as non-spam seem to have resolved themselves; most likely those mails are accurately classified as spam, and the users who reportedly wanted to receive them are not actually noticing when they are blocked.

The effect of our blocking success on the end-user population has been dramatic and overwhelmingly positive, and the spam problem is solved from their perspective. Business has been able to continue without significant interruption. The only user complaints involved the minor number of misclassified mails referenced above. One of the more obvious signs of our success is that many of our users once again consider it odd to see more than a few spams a week in their inboxes, and will even call the support desk to complain when this happens. They continue to appreciate their ability to take some control of the problem when spams do reach them, and many speak of the filter in terms that make it obvious they are able to observe – and have even come to expect – nearly immediate results when they report new spams that have started getting into their inboxes and Bogofilter is trained on them. Periodically we are even contacted by other companies who have received word-of-mouth reports of our success and are interested in the details of our implementation; our users are apparently satisfied enough to mention our success when their contacts complain about spam.

Our ongoing training framework has also worked much better than we anticipated. In theory we should be monitoring all mails that are classified as "unsure" and using these to train Bogofilter. This would be a significant amount of effort, however, so we have instead limited our ongoing training to spam messages users report. Usually several users will report the same message as spam. Training on one message will generally be sufficient to update Bogofilter's opinion of duplicate or similar messages, so the training scripts re-examine each message before it is presented to the administrator. If Bogofilter correctly classifies the message on re-examination it is automatically skipped. On average we only need to train on 15-20 unique messages per day before Bogofilter has "caught up" and the rest of the training queue can be skipped. Using this methodology one administrator is able to process all the incoming requests in four hours per week. The continued success of the filter indicates that at least for the time being this training is sufficient.

So far this solution has also proved entirely scalable, with no obvious ceiling in sight. Mail exchangers can easily be added to support growing mail volume, with each only needing a Bogofilter installation, a copy of the configuration files, and the ability to pull the most recent copy of the wordlists from the spam server once per day. The system meets the goal of being efficient enough to operate in real time without affecting our mail-flow capacity; real-time Bogofilter evaluation of each message costs almost nothing and has not noticeably affected the system load or message processing time (Figure 8). The initial wordlists were 31 MB combined; that has only grown to 40 MB. The spam server is a potential bottleneck both in the need to receive a copy of every incoming message and the ability to lookup cached messages for training in real time, but the single server in place today maintains a CPU load average of 0.1, so this is not any kind of immediate concern. If message lookups become a

```
% ls -s -h random_words.txt
 12M random_words.txt

% < random_words.txt rl | head -c 10240 > words_10K.txt
% < random_words.txt rl | head -c 102400 > words_100K.txt
% < random_words.txt rl | head -c 1024000 > words_1000K.txt
% < random_words.txt rl | head -c 10240000 > words_10000K.txt

% /usr/bin/time sh -c '< words_10K.txt bogofilter'
0.06user 0.03system 0:00.08elapsed 101%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (496major+291minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_100K.txt bogofilter'
0.36user 0.03system 0:00.38elapsed 101%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (498major+440minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_1000K.txt bogofilter'
1.51user 0.10system 0:02.52elapsed 63%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (497major+978minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_10000K.txt bogofilter'
5.74user 0.14system 0:05.88elapsed 99%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (499major+1033minor)pagefaults 0swaps
```

Figure 8: Run times of Bogofilter for various message sizes, using our production wordlists. These times are relative to a dual-CPU 700MHz PIII, the least-powered server in this architecture.

bottleneck we can easily convert the header lookup to a faster database format or something similar. Extremely large organizations may find keeping up with the ongoing training is difficult, as four hours for this environment is potentially a full-time position for an organization with 10 times our mail volume. This can be further automated, however, using methods similar to those we describe in "Future Work" below.

As with any other filtering solution, we have observed spammers attempting to adjust their techniques to get around this filter. However, we have not seen them succeed, as evidenced by our ability to sustain a 98-99% block rate for more than a year. Initial attempts based on weaknesses and bugs in particular early implementations of Bayesian filters ceased to be effective long ago. Wordlist poisoning, token obfuscation, token dilution, and microspam attacks abound, but have had no significant effect on our ability to detect spam messages. This is despite a popular belief that they collectively represent an Achilles' heel of Bayesian filters [EMM, JdeBP2]. As Graham originally predicted, and he and others have since repeated [GRA2, JGC2], the attackers either completely fail to understand the nature of the system they are attacking, or the Bayesian filters simply adapt. Where wordlist attacks have had any success at all it is probable that Effective Size Factor and Bayesian Noise Reduction techniques will be effective in mitigating their value, as has already been demonstrated elsewhere [ROB2, ZDZ]. Microspams, which contain only a URL and a handful of words, show the most current potential for giving the filter trouble; however, each variant works at most once, since the URLs and the unique message headers immediately become high spam indicators, and the lack of other information provides no opportunity for non-spam tokens which could lower the spamicity of the message. In any case this is at best a temporary advantage for spammers, as there are filter enhancements to deal with these as well; some of these are discussed in the next section.

Future Work

Although this implementation is functioning well beyond expectations and has shown no degradation of performance in the time it has been running, there is room for improvement, both to increase filtering efficiency and to stay ahead in the arms race. The point of adaptive filters of course is that they will automatically keep up as spam changes, but there are process efficiencies and theory improvements to incorporate. There are also possible side-channel attacks we need to be wary of.

First, we are planning an upgrade to a more recent version of Bogofilter. We implemented on version 0.13.0. While this has proved stable and reliable in our configuration, it is rather out of date. Newer versions promise increased functionality and configuration options. In addition there are lexer changes to support continuing work in the field of applying Bayesian filtering to spam, such as the way HTML messages are handled. Most importantly, new versions of Bogofilter contain support for Effective Size Factor (ESF), which is designed to account for loss of filter discrimination introduced by naturally occurring volatility in the data as a result of token redundancy.[3]

In addition to upgrading Bogofilter, we would like to track the shifting nature of our spam more rigorously by being more proactive in training on messages which are misclassified or classified as "unsure." As noted above, we currently primarily rely on end users to forward us misclassifications; while this currently works, at the moment it is only obvious to end users when mails that should have been blocked arrive in their inboxes. It is not obvious to them when legitimate mails on certain topics are flagged as "unsure." As the wordlists are populated with more and more tokens that are only spam, we run the risk of introducing false positives.

One proposed method of addressing this issue is to make it visually obvious to end users when messages are classified as "unsure" so that they can forward them to the appropriate training address as they see fit. We do not want to inconvenience them by asking them to install rules to filter mails into separate folders again, so we are investigating ways to modify the way the message displays in Outlook. One option is to use the "X-Message-Flag" header to provide a message indicating the mail is suspected to be spam; Outlook will present this to the user as an alert header above the message body. If users cooperate, this would allow us to train on a higher percentage of unsure mails (especially those

[3]Spammy tokens tend to appear in groups. For example, an email containing the token "mortgage" is more likely to contain the token "mortgage" again, or other related spammy tokens such as "refinance." This token redundancy can have a negative impact on filter discrimination by artificially magnifying the combined probability disproportionately between large and small emails. The good news is that problems with token redundancy can be dealt with mathematically; Gary Robinson has evidence to suggest that applying an Effective Size Factor (ESF) improves filter discrimination significantly by reducing the impact of this redundancy [ROB2]. Greg Louis' work confirms this [LOU6].

Note that token redundancy should not be confused with basic statistical independence, though the issues may appear similar. Confusion about this distinction has led to much ado about the violation of statistical independence in the context of Bayesian mail filtering because current approaches assume statistical independence of the input data where it does not actually exist. For example, training on error violates statistical independence by selecting messages for training based on prior examination, while the algorithms assume that training is done using a truly random sample of messages. Some have claimed this lack of actual independence represents a weakness in the use of Bayesian classifiers that is somehow "exploitable" [ALL]. Violations of assumptions of statistical independence are not news to probability theorists, however, since these assumptions are nearly always violated in practice [ROB3]. In addition, there is work to suggest that even the existence of the assumption of statistical independence has been overstated in the context of Bayesian classifiers [DOM].

which are non-spam) without requiring administrators to do the sorting manually.

However, we would prefer to rely on end-user interaction as little as possible. We intend to implement tools that will allow us to test Bogofilter automatically on incoming mails when we can make reasonable guesses on the status of those messages independently. Probable spam or non-spam messages which Bogofilter apparently misclassifies can be flagged for administrator review, and the filter can be corrected if necessary. Any spam-blocking methods which provide useful categorization of some spam but are either too resource-intensive or have an unacceptably high false-negative rate for use in our production environment are good candidates for this kind of secondary screening. One simple option we will likely implement is to create and advertise a low-priority mail exchanger tarpit; these take advantage of the fact that a fair amount of spam software attempts to target low-priority mail exchangers with the assumption they are less heavily guarded than primary exchangers. Mail sent to this tarpit would be flagged for review, with the assumption that it is all spam. Any mails which Bogofilter was unsure about would be good candidates for training. Another option is to reimplement tools like Vipul's Razor on secondary hardware to scan a random sample of cached incoming mail. For legitimate mails, we will likely begin randomly sampling outgoing mail and flagging any messages which Bogofilter classifies as unsure or spam. Note that none of these tools would be added as active filtering mechanisms, but they could be useful outside of the mail flow in automating the process of keeping Bogofilter's wordlists up to date with the shifting nature of spam.

We will also continue to tweak the filter options as appropriate, and may make some of them dynamic. Possible options here include automatically modifying the Bogofilter configurations on the mail exchangers based on time of day to filter mail more aggressively during high spam periods such as weekends and late night, or at least to flag unsure messages during these times for scanning. Similarly, if the frequency of microspams increases and these become a problem we may lower `min_dev` to catch more of them on the first pass; another option is to make `min_dev` (or other parameters) dynamic based on message size, e.g., set `min_dev` to 0 for messages less than one kilobyte in size. It is entirely possible, however, that the filter will simply adapt; in this scenario things like normal local SMTP headers would become high enough spam indicators that microspams would always be blocked, while both short and long legitimate messages would get through due to their legitimate tokens. In any case, both of these types of dynamic configuration would probably only be necessary temporarily; eventually Bayesian mail classifiers will likely reach the point of using meta data – such as message size, arrival time, number and type of attachments, etc. – as spam/non-spam tokens.

In addition, we have done some investigation to see if we could use a lower base `spam_cutoff` and let even less spam through. Examining the classification of both the spams and legitimate mails that are being classified as "unsure," it looks like we would be safe dropping this value to 0.85 or even lower. However, we will need to do much more rigorous testing before we make such a fundamental change. We will also preferably have the above spot-checking of active mail and other measures in place to track any increase in the ongoing probability of false positives using this cutoff.

We would like to improve the caching system or, preferably, remove the need for it altogether. Storing emails for this kind of retrieval is not ideal. Recreating headers that Exchange has removed is inefficient at best and does not always work. At the least we would like to move to a system where each message can be flagged in a way that Exchange will not modify so that a simplified lookup will work, but this is not likely to be possible without adding something to the message body or requiring users to forward messages in a specific format. Both of these options are at odds with business standards. Other Bayesian classifiers such as DSPAM have experimented with keeping a record in the database of which tokens were present in a given message, but it is questionable whether this would scale to an environment such as ours.

Finally, we will need to reconsider rejecting spams during the SMTP exchange instead of silently dropping them. This gives spammers information about the delivery status of their messages which allows for the possibility of training an "evil" Bayesian filter on what messages our filter rejects or accepts. Given enough messages, this would allow spammers to craft messages specifically designed to pass through our filter. John Graham-Cumming has discussed this attack in detail [JGC]. However, this technique is easily defeated by simply not returning any information to spammers on the delivery status of their messages. This is non-ideal; it means that in that in the case of false positives the legitimate sender would not get an error to indicate their message was not received. However, the low false-positive rate demonstrated by this type of filtering likely justifies this inconvenience.

Related Work

At this time several other medium to large environments are known to be having success monitoring their mail flow with Bogofilter using single, centralized wordlists:

- York University in Toronto recently deployed Bogofilter as a classifier for their environment of 60,000 user accounts. Incoming mail volume is on the order of hundreds of thousands of messages per day. At the time of writing, this implementation was too new to have reliable numbers available, but early results are promising. In this implementation messages pass through Bogofilter

after DCC has already scanned them and rejected spams it can detect; approximately 30-40% of this remaining incoming mail is initially being flagged as spam by Bogofilter. This is an order of magnitude higher than the block rate of the SpamAssassin implementation that Bogofilter replaced, with a dramatically lower rate of false positives reported.

- A large ISP in Australia is using a modified version of Bogofilter with a single wordlist to watch 150,000 mailboxes. Over 1 million messages are processed per day. Bogofilter is believed to be around 95% effective in this environment, with no false positives reported in six months of operation. The wordlist management is completely centralized, with no user input whatsoever. Administrators keep Bogofilter's training current by manually scanning and training on random samplings of 100-300 "unsure" emails per week.

Both of these deployments have followed implementation and maintenance methodologies similar to the ones described in this paper.

Conclusion

Spammers have shown extreme willingness to adapt using any means at their disposal to spread their messages. Assumptions that they will be unwilling or unable to expend considerable resources or break the law to attack their targets are at best unfounded. Blocking techniques aimed at the mail routes and protocols are not directed at immutable properties of spam and therefore seem unlikely to succeed and are likely instead to drive spammers to more and more illicit methods and increasing collateral damage. Paul Graham was correct in noting that the content of spam is the one thing which cannot be changed arbitrarily; it can be altered and obfuscated, but at some point the content still must be there, or there is no potential for any return on the spammers' investment. Content-based filtering is therefore the best currently available approach to the problem, and our results show that Bayesian filters can be viable for the long term as adaptive content filters.

Our results demonstrate that, far from being just another failed attempt at producing a comprehensive and sustainable solution to the spam problem, Bayesian filters can be sufficient without aid from secondary methods. This is true even in large environments with central control, provided these filters are implemented carefully and with a solid understanding of the theory supporting them. There is a tremendous difference throughout the IT field between systems which truly can not stand the test of time and those which are simply difficult or tedious to implement correctly, and it is irresponsible to dismiss something with the latter property as though it had the former. We would all prefer it if we could block spam using easily implemented and foolproof methods, but if complicated solutions are required to win this fight, doing our homework seems the least we can do given the severity of the problem. Bayesian methods deserve further development, testing, and careful consideration before they are dismissed due to false assumptions or incomplete understandings of their value and effectiveness. Organizations and individuals are encouraged to implement similar solutions to the one detailed here and attempt to duplicate and verify our results.

Do we think Bayesian filters are a permanent solution to all spam? No. If nothing else, Moore's Law dictates that brute force attacks will eventually be viable. There is no reason to believe, however, that spammers will bother waiting that long just to implement high-cost attacks. Based on their previous patterns they are much more likely to attempt to sidestep the issue entirely and move to less direct methods: the spam equivalent of side-channel attacks. We are already seeing signs of these, as spammers are using browser malware to inject ads directly into web site content and are increasingly relying on worms to create zombie machines to send their spam for them. At the paranoid extreme, a fairly obvious convergence of these practices would be worms which inject ads directly into outgoing user emails, turning every legitimate mail into spam at the same time. The current malware epidemic demonstrates that such a methodology would be a much more cost-effective practice for spammers than escalating the filter war indefinitely. No filtering technology that only aims to block spam messages would be useful in stopping this type of spam, since any mail blocked would by definition be a false positive. SMTP replacements and authentication schemes would be similarly useless, since these messages would be sent through valid mail routes using appropriate credentials just as many worms are today. If and when we reach this point, however, we will no longer even be dealing with unsolicited commercial and bulk email. We will be dealing with something else. Filtering may be the most effective method for dealing with spam that is simply commercial email messages, but malware is something completely different.

Regardless of the possible future of spam, the most value offered by effective Bayesian filters is to be gained today while spammers are still unable to circumvent them and have not yet determined their next method of attack. The war against spam will no doubt continue to be fought with a range of methods across various platforms. No single technology will likely emerge which is capable of dealing with all attacks equally well. If the next phase does center around malware, it will only be won if we can get ahead of the spammers now and get a handle on the current virus epidemic. If Bayesian filters are at least robust enough to win the filtering battle for the foreseeable future they will be invaluable in buying administrators desperately needed time to move beyond filter maintenance to the much more serious problems of end-user

computer security. Instead of carelessly dismissing these filters as already beaten and wasting time pursuing weaker solutions, administrators should take advantage of the opportunities they provide to go on the offensive and finally gain an advantage over spammers before it is too late.

Availability

Vipul's Razor is available at http://razor.sourceforge.net/ .

Bogofilter is available at http://bogofilter.sourceforge.net/ .

qmail-qfilter is available at http://untroubled.org/qmail-qfilter/ .

Acknowledgements

We thank Paul Graham, Gary Robinson, and others for their pioneering work in this area. We especially thank Greg Louis and David Relson of the Bogofilter project for their responsiveness, feedback, and tuning suggestions throughout our initial training and implementation phases.

Author Information

Jeremy Blosser graduated from Indiana Wesleyan University in 1998 with a BS in political science, history, and philosophy. He moved to Texas to get a real job and has been a Web engineer and occasional system administrator for VHA, Inc. since 2000. He prefers playing with his three kids to sorting spam and can be reached at jblosser@vha.com or jblosser@firinn.org .

Raised by wolves, Dave Josephsen dedicated his life to feeding the naked and clothing the hungry. After a six-year stint in the United States Marine Corps, he became a tech support guru to the stars and eventually found his way to systems administration. He can be found in a server room near you, or contacted via email at djosephs@vha.com or dave@homer.cymry.org .

References

[ALL] Allman, Eric, *The State of Spam*, https://db.usenix.org/events/usenix04/audio/allman.mp3, June 2004.

[BAA] Baard, Mark, *Going Upstream to Fight Spam*, http://www.wired.com/news/print/0,1294,61971,00.html, January 2004.

[BOW] Bowers, Jeremy, *Spam Filtering's Last Stand*, http://www.jerf.org/iri/2002/11/18.html, November 2002.

[DOM] Domingos, Pedro and Michael Pazzani, *Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier*, http://www.cs.washington.edu/homes/pedrod/papers/mlc96.pdf, 1996.

[EMM] Dreyfus, Emmanuel, *Mail-Filtering Techniques*, http://www.onlamp.com/pub/a/onlamp/2004/05/20/mail_filtering.html, May 2004.

[ESR] Raymond, Eric S., *2003 MIT Spam Conference: Lessons from Bogofilter*, http://www.usenix.org/publications/login/2003-06/openpdfs/spam.pdf, January, 2003. *In this talk Raymond explained his rationale for developing Bogofilter; he was primarily interested in putting a tool in the hands of end users since it was believed that Bayesian methods would not work in a centralized fashion. This point was summarized by Chris Devers in the June, 2003 issue of ;login magazine as follows: "As good as Graham's Bayesian algorithm is, ESR felt – as did many of the other speakers – that the nature of your spam/ham corpus is much more significant than the relative difference among any handful of reasonably good algorithms. (Back to the oft-repeated point about how corpus effectiveness falls apart when used for a group of users, as opposed to individuals.)"*

[GRA] Graham, Paul, *A Plan For Spam*, http://www.paulgraham.com/spam.html, August 2002.

[GRA2] Graham, Paul, *So Far, So Good*, http://www.paulgraham.com/sofar.html, August 2003.

[JAC] Jacob, Philip, *The Spam Problem: Moving Beyond RBLs*, http://theory.whirlycott.com/~phil/antispam/rbl-bad/rbl-bad.html, January 2003.

[JdeBP] Pollard, Jonathan de Boyne, *SPF is harmful. Adopt it.*, http://homepages.tesco.net/~J.deBoynePollard/FGA/smtp-spf-is-harmful.html, 2004.

[JdeBP2] Pollard, Jonathan de Boyne, *No anti-UBM measure for SMTP-based Internet mail works*, http://homepages.tesco.net/~J.deBoynePollard/FGA/smtp-anti-ubm-dont-work.html#Bayesian, 2004.

[JGC] Graham-Cumming, John, *2004 MIT Spam Conference: How to beat an adaptive spam filter*, http://www. jgc.org/SpamConference011604.pps, January 2004.

[JGC2] Graham-Cumming, John, *Fooling and poisoning adaptive spam filters*, http://www.sophos.com/sophos/docs/eng/papers/WP_PMFool_US.pdf, November 2003.

[KNO] Knowles, Brad, *Considered Harmful: SPF . . .*, http://bradknowles.typepad.com/considered_harmful/2004/05/spf.html, May 2004.

[LOU] Louis, Greg, *Tuning Bogofilter's Robinson-Fisher Method – a HOWTO*, http://www.bgl.nu/bogofilter/tuning.html, April 2003.

[LOU2] Louis, Greg, *Bogofilter Training: Comparing Full Training with Training-on-error*, http://www.bgl.nu/bogofilter/training.html, April 2003.

[LOU3] Louis, Greg, *Bogofilter Training: Comparing Full Training with Training-on-error, part 2* http://www.bgl.nu/bogofilter/training2.html, April 2003.

[LOU4] Louis, Greg, *Bogofilter Parameters: Effect of varying s and mindev, continued: Comparison of*

results from different email sources, http://www.bgl.nu/bogofilter/smindev3.html, April 2003.

[LOU5] Louis, Greg, *Is Bogofilter Scalable?* http://www.bgl.nu/bogofilter/scale.html, November 2002.

[LOU6] Louis, Greg, *Token Redundancy and the Effective Size Factor*, http://www.bgl.nu/bogofilter/esf.html, May 2004.

[MER] Mertz, David, *Spam Filtering Techniques: Comparing a Half-Dozen Approaches to Eliminating Unwanted Email*, http://gnosis.cx/publish/programming/filtering-spam.html, August 2002.

[PAG] Paganini, Marco, *ASK: Active Spam Killer* http://www.usenix.org/events/usenix03/tech/freenix03/full_papers/paganini/paganini_html/node2.html#SECTION00022000000000000000, April 2003.

[ROB] Robinson, Gary, *Spam Detection,* http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html, September 2002.

[ROB2] Robinson, Gary, *Handling Redundancy in Email Token Probabilities,* http://garyrob.blogs.com/handlingtokenredundancy94.pdf, May 2004.

[ROB3] Robinson, Gary, *Spam Filtering: Training to Exhaustion,* http://www.garyrobinson.net/2004/02/spam_filtering_.html, February 2004.

[SEL] Self, Karsten M., *Challenge-Response Anti-Spam Systems Considered Harmful*, http://linuxmafia.com/faq/Mail/challenge-response.html, December 2003.

[WAR] Ward, Mark, *How to Make Spam Unstoppable*, http://news.bbc.co.uk/1/hi/technology/3458457.stm, February 2004. *This article references John Graham-Cumming's work in defeating Bayesian filters using other Bayesian filters. While Graham-Cumming found this method can work, his conclusion was that it is very costly and quickly blocked. Nevertheless, many administrators and weblogs continue to point to this experiment (and specifically this article) as "proof" that Bayesian filters can be easily defeated.*

[ZDZ] Zdziarski, Jonathan A., *Bayesian Noise Reduction: Progressive Noise Logic for Statistical Language Analysis,* http://www.nuclearelephant.com/projects/dspam/bnr.html, February 2004.

Appendix A

The scripts below are what we used to create our initial wordlists and tune Bogofilter. This functionality can now be found in the scripts that ship with Bogofilter, but these are provided for illustration purposes and in the hope that they may be useful.

```
#!/bin/sh

# retrain.sh: Test various bogofilter.cf values. Create randomized message
# lists, fully train on 10k each spam and non-spam, train on error for other
# messages, test on 5k each spam and non-spam. Repeat until all messages are
# trained on error twice, then output stats for each test. Log everything.

# syntax: retrain.sh <logfiletag>

cd /usr/share/bogofilter/retrain/

echo "removing old wordlists..."
rm -f data/{good,spam}list.db

if [ ! -f list ]; then
        echo "making lists..."
        ./makelists.sh 1>&2
fi

echo "doing 10k spam..."
while read
do
        echo "${REPLY}" 1>&2
        < "${REPLY}" ./bogofilter -C -c ./bogofilter.cf -s
done < 10k_spam_list.random

echo "doing 10k notspam..."
while read
do
        echo "${REPLY}" 1>&2
        < "${REPLY}" ./bogofilter -C -c ./bogofilter.cf -n
done < 10k_notspam_list.random

echo "doing rt round 1..."
./rt.sh rest_list.random | tee ./rt."${1}"-1.log | sed -f ./rt.sed 1>&2

echo "doing tt round 1..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-1.log | sed -f ./rt.sed 1>&2
```

```
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-1.log | sed -f ./rt.sed 1>&2
echo "doing rt round 2..."
./rt.sh rest_list.random | tee ./rt."${1}"-2.log | sed -f ./rt.sed 1>&2

echo "doing tt round 2..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-2.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-2.log | sed -f ./rt.sed 1>&2

echo "adding tt messages round 1..."
./rt.sh 5k_list.random | tee ./rt."${1}"-3.log | sed -f ./rt.sed 1>&2

echo "doing tt round 3..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-3.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-3.log | sed -f ./rt.sed 1>&2

echo "adding tt messages round 2..."
./rt.sh 5k_list.random | tee ./rt."${1}"-4.log | sed -f ./rt.sed 1>&2

echo "doing tt round 4..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-4.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-4.log | sed -f ./rt.sed 1>&2

echo "doing stats..."
echo "1st run:"
./getstats.sh ./tt."${1}"-1.log
echo "2nd run:"
./getstats.sh ./tt."${1}"-2.log
echo "3rd run:"
./getstats.sh ./tt."${1}"-3.log
echo "4th run:"
./getstats.sh ./tt."${1}"-4.log
```

==

```
#!/bin/sh

# makelists.sh: Generate randomized message lists needed by retrain.sh.

# This script requires rl, found here:
# http://tiefighter.et.tudelft.nl/~arthur/rl/

find /var/spam/corpii/{NOTSPAM,SPAM} -type f > list
< list rl > list.random
< list.random grep -i '/corpii/spam' | head -10000 > 10k_spam_list.random
< list.random grep -i '/corpii/notspam' | head -10000 > 10k_notspam_list.random
cat 10k_spam_list.random 10k_notspam_list.random | rl > 10k_list.random
< list.random grep -i '/corpii/spam' | tail -5000 > 5k_spam_list.random
< list.random grep -i '/corpii/notspam' | tail -5000 > 5k_notspam_list.random
cat 5k_spam_list.random 5k_notspam_list.random | rl > 5k_list.random
sort 10k_list.random 5k_list.random | grep -v -F -f - list.random | \
                                                    rl > rest_list.random
```

==

```
#!/bin/zsh

# rt.sh/tt.sh: Process messages specified in provided file through Bogofilter.
# As tt.sh, output Bogofilter's classification.  As rt.sh, output
# classification and train Bogofilter on error.

BOGOFILTER="/usr/share/bogofilter/retrain/bogofilter"
BOGOFILTERCF="/usr/share/bogofilter/retrain/bogofilter.cf"

< "${1}" | while read
do
    printf "%41s: " `echo "${REPLY}" | sed -e 's%/var/spam/corpii/%%'`
    REAL=`echo "${REPLY}" | sed -e 's%/var/spam/corpii/\([^/]\+\)/.*%\1%'`
    printf "%7s: " "${REAL}"
    BOGO=`< "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -v`
    GUESS=`echo "${BOGO}" | sed -e 's/.*\(Spam\|Legit\|Unsure\).*/\1/'`
    SPAMICITY=`echo "${BOGO}" | sed -e 's/.*\(spamicity=[^ ]\+\),.*/\1/'`
    printf "%6s: " "${GUESS}"
    printf "%20s\n" "${SPAMICITY}"
    if [[ "`basename ${0}`" == "rt" ]]; then
        case "${GUESS}" in
            "Spam")
                if [[ "${REAL}" == "NOTSPAM" ]]; then
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -n
                fi
            ;;
            "Legit")
                if [[ "${REAL}" == "SPAM" ]]; then
                  < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -s
                fi
            ;;
            "Unsure")
                if [[ "${REAL}" == "NOTSPAM" ]]; then
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -n
                else
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -s
                fi
            ;;
        esac
    fi
done
```

==

```
# rt.sed: Colorize output of rt.sh.

s/\( SPAM\):/^[[1;31m\1^[[0m:/
s/\(NOTSPAM\):/^[[32m\1^[[0m:/
s/\(Spam\)/^[[1;31m\1^[[0m/g
s/\(Legit\)/^[[32m\1^[[0m/g
s/\(Unsure\)/^[[7;34m\1^[[0m/g
```

==

```
#!/bin/sh

# getstats.sh: Output summary of results from retrain.sh-generated logs.

echo "false positives: `grep 'NOTSPAM:   Spam:' ${1} | wc -l`"
echo "false negatives: `grep ' SPAM:  Legit:' ${1} | wc -l`"
echo "unsure spams   : `grep ' SPAM: Unsure:' ${1} | wc -l`"
echo "unsure notspams: `grep 'NOTSPAM: Unsure:' ${1} | wc -l`"
```

Appendix B

The scripts below are used for filtering our messages during the SMTP exchange.

```
#!/bin/sh

# qq-qfilter: log the mail, then add and fix* the Bogofilter header, then
# forward the mail to the caching server, then check if it's spam and refuse it
# if it is.

# A full description of qmail-qfilter's operation is beyond the scope of this
# paper, but in summary, it takes a mail message on stdin and pipes it through
# a "--" delimited list of filters.  Each filter's stdout becomes stdin for the
# next filter.  Exit codes are checked and manipulated to provide the calling
# qmail daemon what it expects.

# *"Fixing" the Bogofilter header means making sure it is the first line in the
# header.  The version of Bogofilter we use does not place this header
# consistently; this has reportedly been fixed in current versions.

exec /usr/bin/qmail-qfilter /usr/local/bin/qfilter-logger -- \
      /usr/bin/bogofilter -l -e -p -- \
      /bin/sed -e '1,/^X-Bogosity:/{;' -e '/^X-Bogosity:/!{; H; d; };' \
                  '/^X-Bogosity:/{; p; g; D; }; }' -- \
      /usr/local/bin/qfilter-cache -- /usr/local/bin/qfilter-spamcheck
```

```
================================================================================
```

```
#!/bin/sh

# qfilter-spamcheck: read the first line of stdin as an X-Bogosity header; if
# the message is spam, exit 31 to refuse delivery, otherwise pass the message
# through unaltered.

# This script requires rewind, which of part of DJB's serialmail package and
# can be found here:
# http://cr.yp.to/serialmail.html

if head -n 1 | grep -q '^X-Bogosity: Spam,'; then
   exit 31
fi

/usr/local/bin/rewind

cat -
```

DIGIMIMIR: A Tool for Rapid Situation Analysis of Helpdesk and Support E-mail

Nils Einar Eide, Andreas N. Blaafadt, Baard H. Rehn Johansen and Frode Eika Sandnes
– Oslo University College

ABSTRACT

The system administrators of large organizations often receive a large number of e-mails from its users and a substantial amount of effort is devoted to reading and responding to these e-mails. The content of these messages can range from trivial technical questions to complex problem reports. Often these queries can be classified into specific categories, for example reports of a file-system that is full or requests to change the toner in a particular printer. In this project we have experimented with text-mining techniques and developed a tool for automatically classifying user e-mail queries in real-time and pseudo-automatically responding to these requests. Our experimental evaluations suggest that one cannot completely rely on a totally automatic tool for sorting and responding to incoming e-mail. However, it can be a resource-saving compliment to an existing toolset that can increase the support efficiency and quality.

User Dynamics: Problem Statement

Reading and responding to e-mails from disgruntled users in an organization takes up several hours of a system administrator's daily effort. At the Engineering faculty at Oslo University College, with some 1,500 users, system administrators often encounter an inbox with hundreds of messages in the morning when arriving at work. The task of responding to these e-mail messages can be daunting, time-consuming and tedious. Yet, timely and quality replies are immensely important for the individual user in order for the users to fulfill their role in the organization. Technical difficulties and setbacks, that may seem trivial to a system administrator, can be overwhelmingly frustrating and destructive for the user and can be unnecessarily costly for the organization. Further, it is important that anomaly reports from users get the attention of the system administrators early such that corrective and preventive actions can be taken and to minimize the damage and the repercussions of the anomalies. It is therefore important that the system administrators' inboxes are continuously monitored.

Until very recently there have been few general formal training programs for system administrators with the exception of product specific certifications. Yet, the existing training and certification programmes primarily focus on technical issues. User support is usually learned on the job and there are often few official procedures for how to handle user queries. Attempts at ISO-certifying and standardizing procedures in organizations may result in some of these queries being formalized, but often it is superficial bureaucracy that, in terms of timeliness and responsiveness, may actually reduce quality rather than improving it. Consequently, system administrators often develop their individual practices for handling user queries, and often these "common sense" practices work quite well in practice.

Organizations are dependent on a team of system administrators and these administrators can usually be reached through one point-of-call, such as support@fantasticCorp.com. E-mails sent to such an address are commonly forwarded to the entire team of system administrators that are responsible for user queries. Two obvious reasons behind this strategy are that it is easier for the users to remember and the administrator on-duty will receive the message. However, often individuals in an organization know one or more of the system administrators personally and may decide to send a message to this specific system administrator directly. This is (for the reasons stated) an undesirable practice.

The size of the inbox is dependent on a number of factors. Two important factors are when the inbox was last inspected and the occurrence of certain system events. According to queuing theory, e-mails can be modeled as arriving in a stream with a Poisson distribution, and therefore the size of the inbox is approximately proportional to the time passed since it was last read. This phenomenon is something that everyone coming to work in the morning or returning from lunch, meeting or even a business trip can testify. Further, the occurrence of certain events, such as a system failure or anomaly usually ignites a burst of e-mails. For example a printer breakdown on Friday afternoon does not go down well with users struggling to meet their deadlines. Once a system failure or anomaly affects the work carried out by the user, the users often chose to resolve the problem by contacting the system administrators by e-mail, as it may be hard to reach the system administrator by phone. The more users an anomaly affects the more e-mails the system administrator

receives. Occurrences of anomalous events are hard to predict but they themselves may be modeled using a Poisson-like distribution.

The messages from users can be coarsely classified into four categories: a) automatically generated mails from various processes such as cfengine, at, cron, etc., b) unsolicited e-mail and spam, c) general questions and d) urgent questions, notifications and anomaly reports. Automatically generated mail is easily sorted into assigned folders using simple keyword-based filters, and automatic spam filters are getting increasingly good at reducing the amount of junk in the inbox. General questions from users, such as how to do "this or that," are often not urgent and the task of responding to the question can be delayed to a "low-peak" time. Often, users ask similar questions and the system administrator can retrieve an old answer written to a different user at a previous point in time. In this way, the administrator hopefully saves time if this archived message can be recalled with little effort. However, urgent e-mails from users reporting anomalies, faults and strange behaviour in the computer system should be read and dealt with immediately. Many system administrators in small and medium-sized organizations get a "feeling" for how to read and respond to e-mail. The purpose of this work is to assist the system administrator by simplifying e-mail management. DIGIMIMIR, based on text-mining techniques, automatically clusters and categorizes incoming e-mail into related topics and presents the e-mail categories in terms of identifiable and characteristic keywords. System administrators get a clear overview of the inbox contents, and can thus more easily identify urgent matters that need to be resolved at high priority. In addition, DIGIMIMIR can be connected to a reservoir of pre-answered questions such that the most suitable answers to commonly asked questions are found automatically.

Previous Work

Much has been written on helpdesk support [8, 11, 13, 29] and many commercial systems exist, such as E11 and IssueTracker. These systems primarily focus on tracking historic aspects of customer support, maintenance of searchable knowledge bases and the identification of recurring issues. Many of these products target general businesses. GNATS is a system specifically designed for tracking bugs in software and the maintenance of these in databases [28].

Trouble ticketing systems, such as OTRS (Open Ticket Request System), are useful tools for managing large inboxes, which may be handled by several system administrators concurrently. New requests that arrive in the inbox are given a ticket (e.g., a number) and an automatic reply informing the user that the request is received and will be handled. Other requests on the same issue are automatically grouped together with the existing correspondence related to the ticket. The different system administrators therefore have easy access to the entire history of the requests associated with a ticket. The responses of different system administrators can therefore be coordinated.

Expert system based helpdesk systems have also been explored [35], where the staff running the helpdesk is guided through sequence of decision rules to solve the particular difficulties. A problem with expert systems is the nontrivial establishment of the decision rules. Another strategy is case based reasoning which is especially suited at detecting recurring problems [4]. In the spoken domain recurrent neural networks have been used to route helpdesk calls automatically based on utterances [9].

The difficulties of handling large numbers of requests are commonplace in large organizations. Research effort has gone into the automatic retrieval of answers from existing question-answer lists based on queries, such as the VIENA classroom system, which uses lexical similarity [32, 33], the FAQFinder system [1, 20] which uses semantic knowledge and the FAQIQ system which uses case based reasoning [22]. In a different approach [30] a template strategy is used to answer questions based on information in relational databases. Common to all these strategies is that they are based on already existing knowledge bases.

In addition to the distribution of answers it is also necessary to categorize questions. The idea of automatically sorting e-mail into predetermined categories has been examined by several researchers. In one theoretical study [36] web-mining agents are assessed as a means of automatically sorting e-mail using an uncertainty probability based sampling classifier and rough relation learning.

Recently, there has been a huge public interest in the problems associated with spam, and a substantial effort has gone into developing spam-filtering technology [1, 34]. Notably, by far the most efficient strategy is the statistically based nave Bayesian classification [34]. A nave Bayesian system is trained using a large corpus of spam e-mails and non-spam e-mails and a word signature vector is established for both groups. When a new message arrives, its word signature is compared to both that of the spam and the non-spam signatures and the one that yields the best match determines its category. The spam-filtering problem is related to the problem we are addressing in this paper. However, it is different on two major accounts. Firstly, spam-filtering entails classifying messages into two groups, either spam or non-spam. Second, these categories are defined a-priori. However, in document classification there are many (usually more than two) categories and the categories might not be known and therefore have to be established at run-time. Bayesian classification has in fact also been studied in the context of general document classification [24]. One exciting practical development that has occurred in

parallel with the development of DIGIMIMIR is POPFILE, which is a Bayesian based e-mail classification program [12]. POPFILE works directly on POP3 e-mail accounts and uses Bayesian classification to sort incoming messages into predetermined categories. POPFILE as a software product has reached some maturity, but still suffer from major shortcomings – the most notable (at time of writing) is the lack of IMAP support. The IMAP protocol is particularly applicable in the context of automatic mail sorting applications since mail folders can be managed on the server [5], and it is therefore possible to achieve e-mail client independence. Further, POPFILE is designed only to work in supervised mode and is reliant on training data and therefore cannot be deployed in unsupervised situations where new categories are created dynamically on the fly.

Another exciting and controversial new development is the announcement of Google's new e-mail service Gmail [10]. This is a novel strategy in terms of managing e-mail. The idea is that the users get a large area to store their mails and that e-mails rarely have to be deleted. Document classification techniques are therefore used to navigate and search through this huge reservoir of e-mails. This service is not yet available to the general public, but this new thinking with regards to dealing with e-mail management may prove to be useful in terms of support and helpdesk e-mail management.

In fact, the largest success of text mining and document classification and retrieval technologies has been within the areas of web search engines [21, 25] and search engine technologies have developed at a rapid pace over the last few years. However, there is a number of profound differences between the clustering and classification requirements for indexing web pages and handling streams of incoming e-mail. First, the largest difference is probably the volume of text. The web is huge and the size can be exploited to up the quality of the clustering and classification. Personal e-mail collections or an organization's e-mail collection remain small by comparison. Second, indexing of web pages can be done offline and there are no critical time-restrictions on how fast the pages are indexed. Indexing is often done offline and there is a significant turnaround time from when a change is being made on a document on the Internet until it is being reflected in the search results of the search engines. We could perhaps describe this as an index epoch. However, in order for an e-mail sorting system to have a value, the classification and clustering needs to be continuous, instant and real-time. Third, in the indexing of web pages quality is the prime importance, while in the clustering and classification of e-mail messages quality can be traded for speed. Until now, most of the research into text mining and document classification and retrieval primarily focus on clustering and classification quality and they pay less attention to the real-time operational constraints and the incrementally growing document collections (inboxes).

Finally, some system administrators wisely believe that the best support policy is self-reliance and that users should be able to resolve their own problems as much as possible [7] and hence reduce the need for support.

DIGIMIMIR and Text-mining

DIGIMIMIR employs techniques borrowed from web mining (see [3] for a general introduction to web mining) and terminology mining based on text corpora (see for example [15]). DIGIMIMIR takes a set of messages as input and produces a set of message clusters as output, i.e., related messages are clustered together, and unrelated messages are placed in different clusters. The step comprises several phases. First, the system must retrieve the messages, then each message is pre-processed and transformed into a text vector. The set of text vectors representing the set of messages are used as input to the clustering algorithm so as to compute the most suitable clusters and finally the results are presented to the recipient (user).

A dedicated e-mail account is set up and DIGIMIMIR polls the inbox at regular intervals to check for new incoming messages. The inbound messages are processed as follows: Each message is treated as a separate entity and used as a basis for computing a word vector. A word vector represents a set of words as a vector, where each word in a dictionary is assigned a specific position in the vector, thus the size of the vector equals the size of the dictionary used. The presence of a word is marked by a positive non-zero integer, where the value represents the number of times the word occurs in the text. A zero denotes the absence of a specific word. Cleary, different messages containing different words have different word vectors in the word space. Hence the phrases "nuts and bolts" and "bolts and nuts" would both yield the same word vector.

To generate a word vector the text is organized into individual words. The first step is to filter high frequency words, also known as stop words [31, 14]. This is achieved using a stop word dictionary computed from word frequency lists [17]. Next, word stemming is employed to obtain the general form of words [27] with the purpose to reduce the size of the word vectors. Then, a dictionary-based spell checking technique is used. I.e. a reference dictionary comprising of all the possible valid words in the language in all grammatical forms is used. If an entry in the text dictionary cannot be found in the reference wordlist then the entry is tagged as a potential incorrect spelling. All entries marked as potential incorrectly spelled words are crosschecked against the reference wordlist using Metaphone [26]. Metaphone is a technique, inspired by SOUNDEX for matching words based on their English phonetic sound and it is particularly suitable for spell checking applications. Entries with no Metaphone match in the dictionary are

considered special terms. Entries with a metaphone match are most likely incorrectly spelled words (see [19] for an excellent survey of automatic spelling correction techniques). Then, each instance of the remaining words is counted and represented in the word vector. The word vector therefore represents the potentially interesting words that are characteristic of the question [6]. Further, the word vectors can be large and contain large amounts of noise. Research into text-mining often strives to reduce the dimensionality, or size, of the word vectors by the means of some transformation [34]. In DIGIMIMIR a simple word vector reduction technique is employed, namely word masking. For each new vector that is being presented to the system only the words present in the given word vector are considered when computing the distance to the other words in the clusters. I.e. all words that are present in documents to be compared are discarded if they are not also present in the document they are compared to. This mechanism prevents unimportant, and most probably, unrelated words to influence the distance measure. Without dimensionality reduction, or word masking, then the auxiliary words may unnecessarily add to the distance between two word vectors that in practice are quite similar.

The word vectors are clustered using the K-means algorithm – a classic and widely known and effective clustering algorithm (see [6]). In clustering the words are represented as vectors in the word space, and the purpose of the clustering algorithm is to assign word vectors that are similar to the same cluster in the vector space and assign word vectors that are different to different clusters. Two vectors are similar if the distance between the two vectors is small, and two vectors are dissimilar if their distance is large. One popular distance measure is the Eucludean distance. The K-means algorithm works as follows: a set of vectors is to be clustered into K clusters. Initially, the vectors are assigned arbitrarily to the K clusters. Then the mean vector for each cluster is computed. Next, the vectors are reassigned to the cluster to which they are closest and the cluster means are recomputed. The process is repeated until some convergence criteria are met.

Finally, the results of the clustering algorithm are presented to the user as a pre-catalogued set of messages. Each time a new message or a group of messages arrives into the system, the process is repeated incorporating the new messages into the clusters.

Quality Measurements

It is relatively easy to assess the quality of classification tasks when they are applied to a training set, as this is a form of supervised learning, where the categories or clusters are predetermined. One can simply compute the success rate as the number of messages that are correctly assigned a cluster, and the error rate as the number of messages that are incorrectly assigned. However, assessing the quality of clustering is more difficult as there is no given mapping between the training category and the assigned cluster. We therefore deployed the following strategy: Each document d_i belongs to a training category c_i and after the algorithm is deployed it is assigned a cluster k_i. After training a category-to-cluster matrix is established where the columns represent the training categories and the rows represent the assigned clusters. An element at column i and row j denotes the number of documents from category c_i that has been assigned cluster k_j.

The quality measures are then computed in three steps. First, for each category the entire column is scanned for the largest value and this is marked as a category-to-cluster mapping. Second, for each cluster the entire row is scanned for the largest value which again is marked as a category-to-cluster mapping. If other elements in the row are also marked as a category-to-cluster mapping (in the first step) then these are remarked as "undecided." Third, the three quantities are computed as follows: the success rate is computed by summing all elements marked as category-to-clusters and dividing by the total number of documents, the failure rate is the sum of all non-zero unmarked elements divided by the total number of documents and finally the ratio of ambiguous messages is the sum of all elements marked as "undecided" divided by the total number of documents.

Test Suites

In order to assess and document the effectiveness of the system through a repeatable experiment, one is dependent on a test suite with pre-categorized messages. The Reuters-21578 Text Categorization Collection [23] is a well-known and widely cited test suite, comprising of Reuters news articles from 1987 that have been classified and indexed by experts and later made available for research purposes. These news articles comprise medium to long, well written, pieces of English text. A news report is long compared to e-mail messages that often are short and poorly written with spelling mistakes and various abbreviations. The Reuters collection is therefore not completely representative of the problem domain. Further, we were unable to use this test suite as the current implementation of DIGIMIMIR is optimized for Norwegian. To manually categorize messages is time-consuming, difficult and error-prone. We therefore deployed three strategies for obtaining test-suites:

First, a small hand crafted test suite, comprising of 100 messages, was used for early testing. This set contains manually categorized fictitious messages, characterized as being easy to cluster and classify.

Second, the students at Oslo University College were used to create the second set of messages from an online "quiz," comprising 160 messages. Four themes with 10 entries each were created. Each entry comprised a picture and a statement, such as a picture of well-known politicians or some hi-tech device. The

students were asked to submit a question related to the entry via a web form. Thus the questions received could therefore be tagged with the given category.

Third, a set of messages from our local UNIX system administrator was collected and manually organized into natural categories and labeled. Unfortunately, this test set was small and contained a diverse set of messages, since the system administrator limited the selection (out of concern for privacy and security) and considered the task low priority. Secondly, as this paper was finalized during late spring early summer, there was not that much traffic as the peak usually occurs in the autumn when new students enroll onto the college courses and acquire accounts on our computer system.

Implementation Details

The implementation consists of the following highly configurable Java components: Controller-module: Controls the flow of information through the system as the modules can be interconnected in an arbitrary manner, i.e., the output of one module can sent to the input of another module etc. Messages are processed as they travel through the various modules on their path to their destination. A typical message-path comprises an input-module, a set of pre-parsers, a parser-module, a set of filters, a sorter-module and an output-module. Multiple instances of a module can be created and used in different parts of the message chain.

The modules are glued together by the means of RMI (Remote Method Invocation) and a global configuration file, allowing various modules to reside on different machines in a distributed manner. This feature allows systems with a high degree of interactivity to be configured, i.e., the system can be configured to provide immediate feedback to instant inbound messages. A special message-path configuration is required, which does not contain conventional input and output modules. This implementation also allows modules to be interconnected to form a tree-structure where all non-leaf nodes are routers that forward messages to yet other modules. For instance, a controller can be configured to identify the language of a message and forward the message to the corresponding controller for the target language of that message. Messages can be rejected and re-routed to different branches of the graph if problems are detected by modules higher up in the tree.

Input-modules retrieve and convert external data into the internal representation used by the system. The current set of input modules can read POP3 (Post Office Protocol) and IMAP (Internet Mail Access Protocol) mailboxes, and external JDBC-compliant database tables provided the correct fields are appropriately configured.

Preparsers modify the incoming messages before they are tokenised into sentences and words. Removal of HTML-tags is a common pre-parser filter operation.

Parsers tokenise texts into sentences and words. A message must be parsed before further processing can take place.

Filters alter or remove words or phrases from messages. For example, a spellchecker replaces incorrectly spelled words with their corresponding correctly spelled words, while a stop-word filter removes words that are not relevant to content of the message.

Sorter-modules attach answers to incoming messages, or signals to the controller that there are no matching answers to the incoming question.

Output-modules return messages back to their source provided they belong to the same category. For example, POP3 and IMAP input messages are returned via the SMTP (Simple Mail Transfer Protocol) output module. Further, the database input messages can be returned to another external JDBC-compliant database using references created by the input module.

GUI (Graphical User Interface)-controllers are equipped with swing-based GUI components which allows for direct user-controller-interaction. The current GUI controller can be connected to multiple controllers simultaneously. A HTML-based web interface is not currently provided, but is planned for a future release. The application is completely written in Java, currently using MySQL 3.23 for persistent storage via Hibernate. The application is developed and tested using Sun's JRE 1.4.2 (Java Runtime Environment).

Results

Figures 1, 2, 3, and 4 show the results obtained running DIGIMIMIR on two datasets. The figures illustrate the accumulative performance of the system, i.e., how the system state changes as messages are added to the system. The horizontal axes indicate the number of messages in the system and the vertical axes represent percentage correct answers, incorrect answers, ambiguous or uncertain answers and new clusters. Figure 1 shows the results obtained using the quiz dataset in unsupervised mode (160 messages), i.e., without training, and Figure 2 shows the results using the quiz dataset in supervised mode (80 messages), i.e., with training. One half of the dataset was used for training and the other half was used during testing. Figure 3 shows the handcrafted dataset in unsupervised mode and Figure 4 shows the hand-crafted dataset in supervised mode. The messages were shuffled into pseudo random order for all the test runs.

All the figures show similar trends, namely that the success rate, the error rate and the percentage of new clusters converge as more messages are added to the system – and this is adhering to expectations. The quiz data reveals a clear difference between the unsupervised and the supervised runs. In unsupervised mode we achieve a successful classification rate of nearly 50% and an error rate of just below 40%. The

percentage of ambiguous or uncertain messages is converging quickly to approximately 15%. Further, the probability percentage of generating a new cluster converges early at just over 10%. The results achieved in supervised mode are nearly twice as good. First, the success rate converges at around 80%, which is nearly a doubling in quality compared to unsupervised mode. This result is consistent with similar experiments reported in the literature on document classification. Second, the failure-rate is converging at 20%, which again is a halving in the number of incorrectly classified messages compared to unsupervised mode. Further, the rate of ambiguous or uncertain messages converges early at around 10%. It is also interesting to observe that the probability of generating a new cluster is converging much slower in supervised mode compared to unsupervised mode and that it is converging at a higher value of approximately 20% compared to 10% for unsupervised mode.

The shape of the graphs in the figures smoothly either decrease (error-rate and clustering probability) or increase (success-rate), but at certain points there

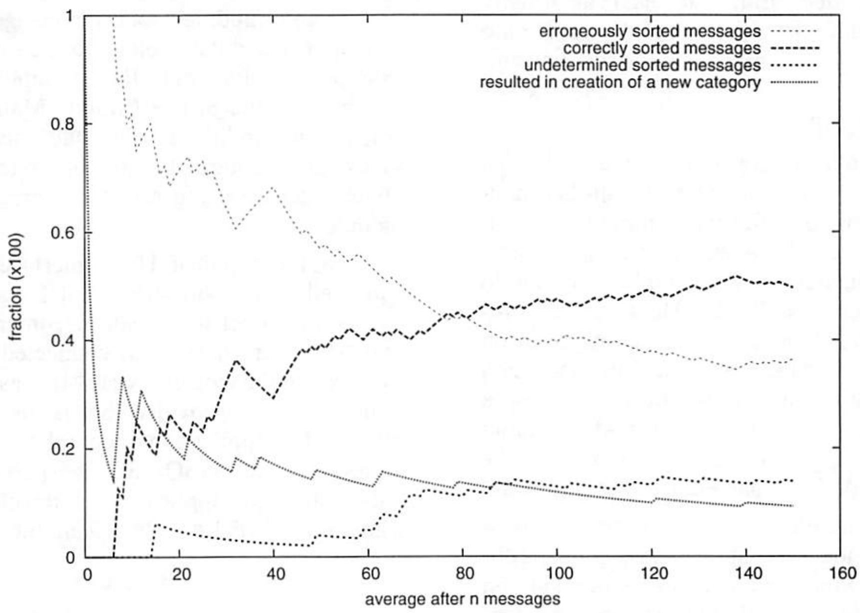


Figure 1: Cumulative success rates, error rates, uncertainty rates and clustering probabilities using the quiz test suite in unsupervised mode.

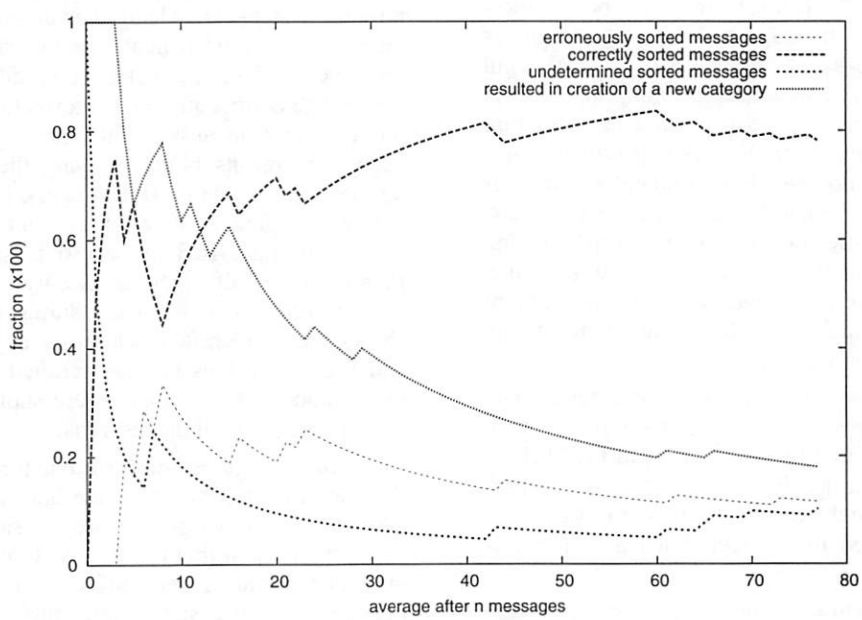


Figure 2: Cumulative success rates, error rates, uncertainty rates and clustering probabilities using the quiz test suite in supervised mode.

appear to be discontinuities in the graphs resulting in temporal setbacks. These discontinuities mark the arrival of very dissimilar messages that result in new clusters being established. When new clusters are added the classification landscape is altered and leads to a temporary classification instability and slightly lower success rates. These messages can be genuine and naturally belonging in new clusters or they may simply be irrelevant or noisy messages.

Similar observations can be made in Figures 3 and 4. Figure 3 shows the accumulative success, error and clustering rates for the custom-made test data in unsupervised mode with very high success rates, and Figure 4 shows the same dataset in supervised mode.

Operational Issues

It would seem natural to integrate DIGIMIMIR with a trouble ticket system such as OTRS. In fact, a trouble ticketing system could be a good front end to DIGIMIMIR. At the entry point all incoming messages are passed through a spam-filter, to remove noise in the input stream, and then the e-mails are

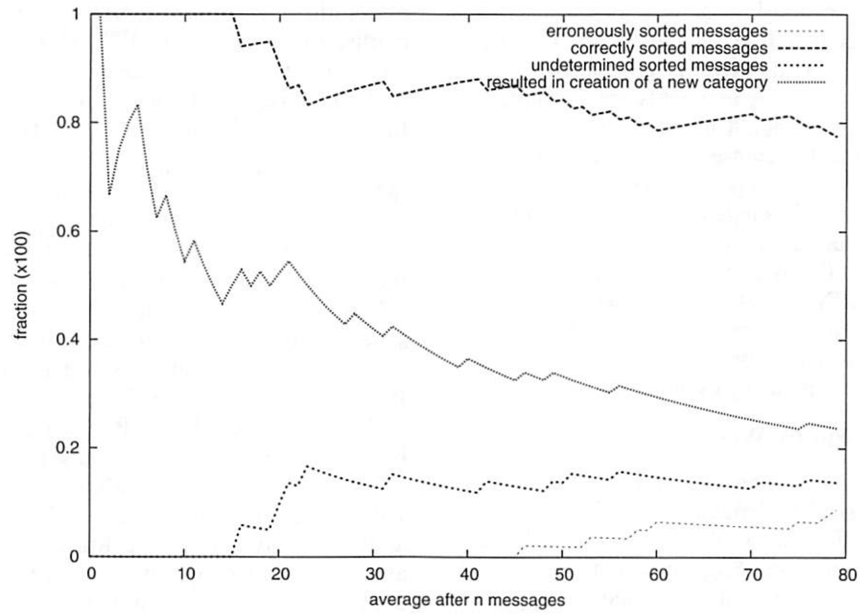


Figure 3: Cumulative success rates, error rates, uncertainty rates and clustering probabilities using the hand crafted test suite in unsupervised mode.

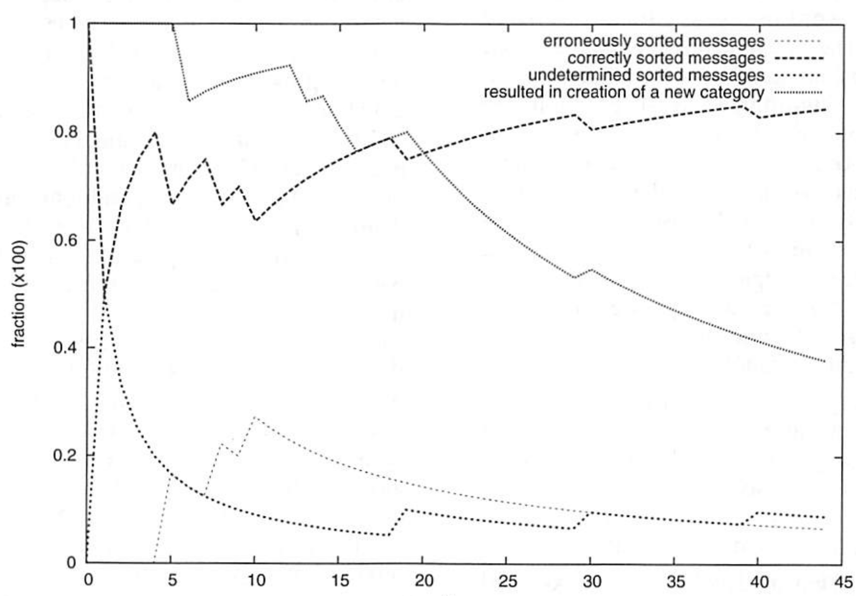


Figure 4: Cumulative success rates, error rates, uncertainty rates and clustering probabilities using the hand crafted quiz test suite in supervised mode.

directly handled by DIGIMIMIR. DIGIMIMIR would immediately classify the message and identify if there are any suitable categories, i.e., a match with a high probability of belonging to the particular cluster. If there is a good match then DIGIMIMIR would respond with a auto-reply if one exists in the system. If one of the tickets associated with a message in the cluster is assigned a response, then that response is also returned to the client. If there is no suitable response, or the probability is below a given threshold then the message is assigned a cluster and a standard ticket notice is returned to the user. These unprocessed messages are later addressed by a second line of system administrators. Each cluster is therefore analogous to a work queue, and once one general answer is generated for the messages in the cluster, all the recipients are forwarded the general reply. There should also be room to give specific and independent responses to a particular ticket in a queue and mark it as such. This would prevent a specific answer to be used as a general answer for automatic distribution. Further, is should also be easy to move a ticket manually to a different cluster if it is detected that DIGIMIMIR has misclassified the message. Further, it should be possible to manually establish new clusters and manually combine existing clusters that have been automatically established.

Future Work

There are many improvements that can be made to DIGIMIMIR. Firstly, the language specific modules need to be internationalized. An absolute medium addition is support for the English language. Ultimately, the tools should be easily extended to support any western language, such that it can be configured using only a standard wordlist for the language, for example the wordlists found on most Unix/linux-type systems, and a stop word list. These are all relatively easily accessible. The main challenge is the word stemming algorithms that must be tailor made for each language. Stemming algorithms have been published for most western languages, but one needs to know the language at least in order to evaluate the effectiveness of these algorithms. Another challenge is pictographic-based scripts such as Chinese. Chinese computer lingo is a good mix of Chinese characters and English terms, at least judging from Chinese Computer Magazines. There is a vast literature on Chinese language processing, and some answers may be found there. POPFILE claims to handle Chinese messages.

Further, we only had the opportunity to experiment with a few clustering techniques. Document classification is an ongoing research topic and better algorithms are continuously published. And the DIGIMIMIR tools will most certainly benefit from improved clustering and classification strategies.

A crucial next step of development is extended IMAP support such that the folders facilities on IMAP servers can be exploited. In addition to polling messages off the IMAP server, DIGIMIMIR should use the messages in the existing folders on the server during the classification and consequently move messages from the inbox to the correct folders on the server. When new clusters are established, DIGIMIMIR should create new folder on the server too. This will allow standard IMAP clients, such as Thunderbird, to be used as front ends to DIGIMIMIR and users would require no additional training. The users will see the new messages in the respective folders and the newly established folders. Further, by manually moving messages from one folder to another, the user can correct the classification engine and manually affect the classification and clustering. This would abolish the need for the DIGIMIMIR GUI component. However, we have had experiences with IMAP clients that do not register folder changes on the IMAP server that are made by other IMAP clients.

A crucial next step of development is extended IMAP support such that the folders facilities on IMAP servers can be exploited. In addition to polling messages off the IMAP server, DIGIMIMIR should use the messages in the existing folders on the server during the classification and consequently move messages from the inbox to the correct folders on the server. When new clusters are established, DIGIMIMIR should create new folder on the server too. This will allow standard IMAP clients, such as Thunderbird, to be used as front ends to DIGIMIMIR and users would require no additional training. The users will see the new messages in the respective folders and the newly established folders. Further, by manually moving messages from one folder to another, the user can correct the classification engine and manually affect the classification and clustering. This would abolish the need for the DIGIMIMIR GUI component. However, we have had experiences with IMAP clients that do not register folder changes on the IMAP server that are made by other IMAP clients. Another long-term improvement would be to introduce the idea of collaborative DIGIMIMIR systems, perhaps via a central body or directory, analogous to how virus filters update themselves on a regular basis. Many of the problems faced by system administrators are independently of organization, but rather dependent on a particular version of a product, for example a security patch for the apache web-server. As system administrators encounter problems and manually establishes categories for these problems, the information can then be shared with other DIGIMIMIR clients via one or more central word-vector reservoirs. Then, when other system administrators encounter similar problems at a later date, they can benefit from the work already carried out by the first system administrator and automatically obtain the new category, perhaps even with a suggested response. However, there are obvious privacy and quality issues that need to be addressed in order to deploy such a strategy.

Another, interesting possibility would be to integrate the system with a network monitoring and

alerting system such as IPSentry. Such system often has many channels of notification including e-mail. Obviously, a message from such a system is very easily detected by DIGIMIMIR and is correctly classified as such a system produces messages with uniform wording and format. More interestingly, notifications from network monitoring and alerting systems could be correlated with user messages. This could greatly help a system administrator more easily assess the impact of a given network anomaly.

Finally, DIGIMIMIR could benefit from a thorough review with regards to efficiency. Currently, the system has been tested with hundreds of messages with no apparent performance bottlenecks. However, large organization could easily receive a thousand messages each day, and maybe keep millions of messages on record. There are no apparent time or space complexity issues that prevent the system from scaling. The major bottleneck of the system is the k-means clustering algorithm, which has a linear time-complexity with respect to the number of messages ($O(c\ k\ d\ n)$, where k is the k-value, d is the number of dimensions for the documents, n is the number of documents and c is the number of iterations required) [16]. Further, the distributed nature of the DIGIMIMIR framework allows it to be configured to run in a distributed manner across a network of workstations, such that the inherent parallelism can be exploited.

Implications of Automated E-mail Support

The deployment of automatic document classification technology in the context of sorting incoming e-mails and automatically providing answers must be done with care. Nearly all requests are unique and the quality of the responses is best maintained by handling the requests manually. However, the quality of a support service is a tradeoff between the quality of the response content and its timeliness. A support department that does not respond to requests or the responses are answered weeks after they were originally sent can be frustrating to the users as they do not feel that their request is taken seriously. On the other hand, a rapid meaningless or obviously auto-generated incorrect reply can be equally frustrating and infuriating to the user and embarrassing to the organization. This may result in aggressive users. It is very difficult to make a foolproof system, i.e., a system that never incorrectly classifies a message and respond with the answer to a different question. To minimize the damage one can adopt psychological techniques, such as using humble wording in the responses. For instance, in DIGIMIMIR we used the following careful wording in the auto-generated replies: "We found that your question X is very similar to question Y. One answer to question Y is Z. Note that this response is automatically generated and a human will evaluate this response hopefully quite soon."

The optimal policy is probably a mixture of manual responses and automated responses. Manual responses should be used during low-peak periods for messages with a lower classification probability, while the automatic response mechanisms are to be used during peak hours when there is not sufficient manual resources to handle the stream of incoming messages. The administrator can then later inspect the requests that arrived during the peak time and assess the responses, and then send corrections to users when appropriate. Such a post-peak period inspection is still more efficient than having to respond to every message manually. It will in some situations be easier to spot a message that is out-of-place when it is placed together with other messages that are related. Ultimately, the inspection cycle is necessary as the 10 to 20% messages are incorrectly classified and needs to be handled manually. If one receives 500 messages a day, then this will account to 50 messages, and if the organization receives 5000 messages a day, this will obviously account to as much as 500 messages.

Politically speaking, an automated system is desirable as it is resource-saving. As long as the financial gains of deploying such a system are larger than the negative impacts of the errors introduced, an organization is likely to embrace such technology. A consequence of this is that in the next instance the system administrators may be budgeted with even fewer resources by the decision makers.

Availability

DIGIMIMIR is constantly under development and is released under a GPL license. Its binaries, source code and documentation can be downloaded from http://www.digimimir.org/.

Conclusions

In this paper we have addressed the problem of e-mail helpdesk support. Text-mining techniques were explored as means of partially automating the support tasks and a tool dedicated to this task was presented. Our experiments confirm that it is possible to achieve 50% or more correctly classified messages in unsupervised mode and 80% correctly classified messages in supervised mode. This can greatly help support staff reduce their workload, especially when combined with an auto-response feature. In operation, the system can exploit a mixture of supervised and unsupervised mode. Messages that are manually approved or classified messages can be used in a supervised manner, while new clusters can be established dynamically and unsupervised in order to get clear overview reports of totally new situations. We believe that document classification technology to a greater extent will be incorporated into the broad range of e-mail handling systems in the years to come due to the great potential for reducing the workload, but to ensure quality there should be a human in the loop. Further, such technology could also help reduce the emergency response time of a support team as emergency messages can be

more quickly identified and separated from less urgent requests.

Acknowledgments

The authors are grateful to the system administrator Sigmund Straumsnes of the Engineering faculty at Oslo University College for providing sysadmin related e-mails. Further, the authors are grateful to the staff and students at Oslo University College who participated in the quiz, and H-L Jian of National Cheng Kung University who provided some of the text mining papers. Finally, the authors acknowledge the useful comments of the anonymous reviewers and Jeffrey Sheldon of the Mathematics Department at Louisiana State University, who has greatly helped us improve the quality of this manuscript.

Author Information

Nils Einar Eide has recently graduated with a B.Sc. degree from the Computer Science Department at Oslo University College (2004).

Andreas Nordeng Blaafladt is an undergraduate student at Oslo University College. He is also a consultant with experience in Unix and network administration.

Baard H. Rehn Johansen is currently working towards a Masters degree at the University of Oslo, Department of Informatics. He received his B.Sc. from the Oslo University College in 2004. Reach him electronically at baard@rehn.no .

Frode Eika Sandnes received a B.Sc. in computer science from The University of Newcastle Upon Tyne, U. K. in 1993 and a Ph.D. in computer science from The University of Reading, U. K. in 1997. Dr. Sandnes research interests include parallel processing and especially multiprocessor taskgraph scheduling, human interaction with mobile devices, error correction codes and system administrations research. He is currently an Associate Professor at the Department of Computer Science at Oslo University College. Reach him electronically at frodes@iu.hio.no .

References

[1] Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An Evaluation of Naive Bayesian Anti-Spam Filtering,' in *Proc. of the Workshop on Machine Learning in the New Information Age*, 2000.

[2] Burke, R., K. Hammond, V. Kulyukin, S. Lytinen, N. Tumuro and S. Schenberg, "Natural Language Processing in the FAQ Finder System: Results and Prospects," in *Working Notes from AAAI Spring Symposium on NLP on the WWW*, pp. 17-26, 1997.

[3] Chakrabarti, S. *Mining the Web: Analysis of Hypertext and Semi Structured Data,* Morgan Kaufmann, 2002.

[4] Chang, K. H., R. Raman, W. H. Carlisle, and J. H. Cross, "A self-improving helpdesk service system using case-based reasoning techniques," *Computers in Industry*, Vol. 30, Num. 2, pp. 113-125, 1996.

[5] Crispin, M., "Internet Message Access Protocol – Version 4rev1," *RFC 3501*, http://www.imap.org/ .

[6] Dunham, M. H., "DATA MINING: Introductory and Advanced Topics," pp 140-142, Prentice Hall, 2002.

[7] Elling, R. and M. Long, "User-setup: A System for Custom Configuration of User Environments, or Helping Users Help Themselves," *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, p. 215, USENIX Association, Berkeley, CA, 1992.

[8] Foo, S., S. C. Hui, and P. C. Leong, "Web-based intelligent Helpdesk-Support Environment," *International Journal of Systems Science*, Vol. 33, Num. 6, pp. 389-402, 2002.

[9] Garfield, S. S. and Wermter, "Recurrent Neural Learning for Helpdesk Call Routing," *Lecture Notes in Computer Science*, Num. 2415, pp. 296-302, 2002.

[10] Google.com, "Welcome to Gmail," http://gmail.google.com/, 2004.

[11] Govindarajulu, C., "The status of helpdesk support," *Communications of the ACM*, Vol. 45, Num. 1, pp. 97-100, 2002.

[12] Graham-Cumming, J., "POPFILE Trust the Octopus," http://popfile.sourceforge.net/ .

[13] Guy, T., "Backstage at the Helpdesk," *Proceedings of the 1999 ACM User Servive Conference,* ACM press, pp. 225-227, 1999.

[14] Ho, T. K., "Fast Identification of Stop Words for Font Learning and Keyword Spotting," *Proceedings of the Fifthth Int'l Conference on Document Analysis and Recognition*, pp. 333-336, 1999.

[15] Justeson, J. & S. Katz, "Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text," *Natural Language Engineering*, Vol. 1, Num. 1, pp. 9-27, 1995.

[16] Kantabutra, S. and A. Couch, "Parallel K-Means Clustering Algorithms on NOWs," *Nectec Technical Journal*, Vol. 1, Num. 6, pp. 243-248, Thailand, 2000.

[17] Kilgarriff, A., "Putting Frequencies in the Dictionary," *International Journal of Lexicography*, Vol. 10, Num. 2, pp. 135-155, 1997.

[18] Kilgarriff, A., "Which Words are Particularly Characteristic of a Text? A Survey of Statistical Approaches," *Proceedings of AISB Workshop on Language Engineering for Document Analysis and Recognition*, Sussex University, pp. 33-40, 1996.

[19] Kukich, K., "Techniques for Automatically Correcting Words in Text," *ACM Computing Surveys*, Vol. 24, Num. 4, pp. 377-439, 1992.

[20] Kulyukin, V. A., K. J. Hammond, and R. D. Burke, "An Interactive and Collaborative Approach To Answering Questions for an Organization, Technical report TR-97-14," Dept. Computer Science, University of Chicago, 1997.

[21] Lai, Y. S., K. A. Fung, and C-H. Wu, "FAQ Mining via List Detection," *Proceedings of the COLING Post-Conference Workshop on Multilingual Summarization and Question Answering*, Taipei, Taiwan, 2002.

[22] Lenz, M. and H. D. Burkhard, "CBR for Document Retrieval – The FAllQ Project," Case-Based Reasoning Research and Development (ICCBR-97), *Lecture Notes in Artificial Intelligence, Num. 1266*, pp. 84-93, Springer-Verlag, Berlin, 1997.

[23] Lewis, D. D., "Reuters-21578 Text Categorization Test Collection Distribution 1.0," http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html, 1997.

[24] Nigam K., A. K. McCallum, S. Thrun, and T. Mitchell, "Text Classification from Labeled and Unlabeled Documents using EM," *Machine Learning*, Vol. 39 Num. 2-3, pp. 103-134, May 2000.

[25] Moldovan, D. and A. Novischi, "Lexical Chains for Question Answering," *Proceedings of COLING 2002 International Conference on Computational Linguistics*, Taipei, Taiwan, 2002.

[26] Phillips, L., "Hanging on the Metaphone," *Computer Language*, Vol. 7, Num. 12, pp. 39-43, 1990.

[27] Porter, M. F., "An Algorithm for Suffix Stripping," *Program*, Vol. 14, Num. 3, pp. 130-137, 1980.

[28] Salvadori, L., "GNATS Revisited," *Sys Admin: The Journal for UNIX Systems Administrators*, Vol. 6, Num. 8, pp. 53-55, 1997.

[29] Shalhoup, R., "A Web-Based Helpdesk," *Sys Admin: The Journal for UNIX Systems Administrators*, Vol. 6, Num. 9, pp. 41, 42, 44-46, 1997.

[30] Sneiders, E., *Automated Questioning Answering: Template Based Approach*, Ph.D. Thesis, Dept. Computer and System Science, Stockholm University, 2002.

[31] Wilbur, J. W. and K. Sirotkin, "The Automatic Identification of Stop Words," *Journal of the American Society for Information Science*, Vol. 18, pp. 45-55, 1992.

[32] Winwarter, W., "Collaborative Hypermedia Education with the VIENA Classroom System," *Proc. First Australasian Conf. on Computer Science Education, ACSE'96*, pp. 337-343, 1996.

[33] Winwarter, W., O. Kagawa, and Y. Kambayashi, "Applying Language Engineering Techniques to the Question Support Facilities in VIENA Classroom," *Database and Expert Systems Applications,* pp. 613-622, 1996.

[34] Yerazunis, W. S., "The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It," *Proceedings of the 2004 MIT Spam Conference*, January, 2004.

[35] Zhao, M., C. Leckie, and C. Rowles, "An Interactive Fault Diagnosis Expert System for a Helpdesk Application," *Expert Systems*, Vol. 13, Num. 3, 203-217, 1996.

[36] Zhong, N., T. Matunaga, and C. N. Liu, "A Text Mining Agents Based Architecture for Personal E-mail Filtering and Management," *Intelligent Data Engineering and Automated Learning – IDEAL 2002*, Lectures Notes in Computer Science 2412, pp. 329-336, 2002.

Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management

Yi-Min Wang, Roussi Roussev, Chad Verbowski, Aaron Johnson – Microsoft Research
Ming-Wei Wu, Yennun Huang, and Sy-Yen Kuo – National Taiwan University

ABSTRACT

Spyware is a rapidly spreading problem for PC users causing significant impact on system stability and privacy concerns. It attaches to extensibility points in the system to ensure the spyware will be instantiated when the system starts. Users may willingly install free versions of software containing spyware as an alternative to paying for it. Traditional anti-virus techniques are less effective in this scenario because they lack the context to decide if the spyware should be removed.

In this paper, we introduce Auto-Start Extensibility Points (ASEPs) as the key concept for modeling the spyware problem. By monitoring and grouping "hooking" operations made to the ASEPs, our Gatekeeper solution complements the traditional signature-based approach and provides a comprehensive framework for spyware management. We present ASEP hooking statistics for 120 real-world spyware programs. We also describe several techniques for discovering new ASEPs to further enhance the effectiveness of our solution.

Introduction

Spyware is a generic term referring to a class of software programs that track and report computer users' behavior for marketing or illegal purposes. In addition, spyware may actively push advertisements to the user by popping up windows, and change the Web browser start page, search page, and bookmark settings. Spyware often silently communicates with servers over the Internet to report collected user information, and may also receive commands to install additional software on the user's machine. Users infected with spyware commonly experience severely degraded reliability and performance such as increased boot time, sluggish feel, and frequent application crashes. Reliability data shows that spyware programs account for fifty percent of the overall crash reports [FTC04]. Saroiu, et al. [SGL04] point out security problems caused by vulnerabilities in spyware programs. A recent study based on scanning more than one million machines show the alarming prevalence of spyware: an average of four to five spyware programs (excluding Web browser cookies) were running on each computer [E04A, E04B].

Current anti-spyware solutions [AA, SB] are primarily based on the signature approach used by anti-virus software: each spyware installation is investigated to determine its file and Registry signatures for use by scanner software to detect spyware instances. This approach has several problems.

First, many spyware programs may be considered "legitimate" in the following sense: their companies sponsor popular freeware to leverage their installation base; since users agree to an End User Licensing Agreement (EULA) when they install freeware, removing the bundled spyware may violate this agreement. In many cases, the freeware ensures the spyware is running on the user's system by refusing to run if its bundled spyware is removed.

Second, the effectiveness relies on completeness of the signature database for known spyware. Beyond the difficulty of manually locating and cataloging new spyware, this approach is further complicated because spyware are full-fledged applications that are generally much more powerful than the average virus [C04], and can actively take measures to avoid detection and removal. Companies creating spyware generate revenue based on the prevalence of their applications and therefore have a financial incentive to create technologies that make it hard to detect and remove their software. They have the need and the resources to invest in developing sophisticated morphing behavior.

Third, some spyware installations may contain common library files that non-spyware applications use. If care is not taken to remove these files from the spyware signatures, scanners using these signatures may break non-spyware applications.

Finally, popular spyware removal programs are commonly invoked on-demand or periodically, long after the spyware installation. This allows the spyware to collect private information and makes it difficult to determine when the spyware was installed and where it came from. A monitoring service that catches spyware at installation time is essential for reducing exposure and avoiding re-infection.

To complement the signature-based approach, we introduce the concept of *Auto-Start Extensibility Points* (ASEPs) as the key to spyware management. Our work

is based on the observation that, in order to monitor users' behavior on an ongoing basis and to maximize the time window for monitoring, an overwhelming majority of spyware programs infect systems in such a way that they are automatically started upon reboot and the launch of most commonly used applications. We use the term ASEPs to refer to the subset of OS and application extensibility points that can be "*hooked*" to enable auto-starting of programs without explicit user invocation. An ASEP may accept one or more ASEP *hooks*, each of which is associated with an auto-start program.

We distinguish two types of ASEP hooking: (1) as *a standalone application that is automatically run* by registering as an OS auto-start extension such as a Windows NT service or a Unix daemon; or (2) as *an extension to an existing application* that is either automatically run (such as *WinLogon.exe* with its *Notify* extensions) or popular and commonly run by users (such as the Internet Explorer browser with its Toolbar extensions).

Figure 1 depicts a Windows-based systems with three layers of gates. The Outer Gates are the entrance points for program files from the Internet to get on user machines. The Middle Gates are the ASEPs that allow programs to hook a system to essentially become "part of the system" from a user's point of view. The Inner Gates control the instantiation of program files. Our solution, named *Gatekeeper*, identifies and monitors the Middle Gates and exposes all ASEP hooks to allow effective management of spyware.

Problem Formulation and Decomposition

Figure 2 illustrates the "life cycle" of the spyware management process and provides a problem decomposition that enables us to systematically reason about this complex problem. Note that our current solution does not address the issues of malicious software such as RootKits [P03]; we will briefly discuss malicious behavior in the Discussions section.

In Step (1), given a spyware-infected machine, since we do not have sufficient context information for already-installed spyware programs, we rely on the signature-based scanning and removal tool (such as Ad-

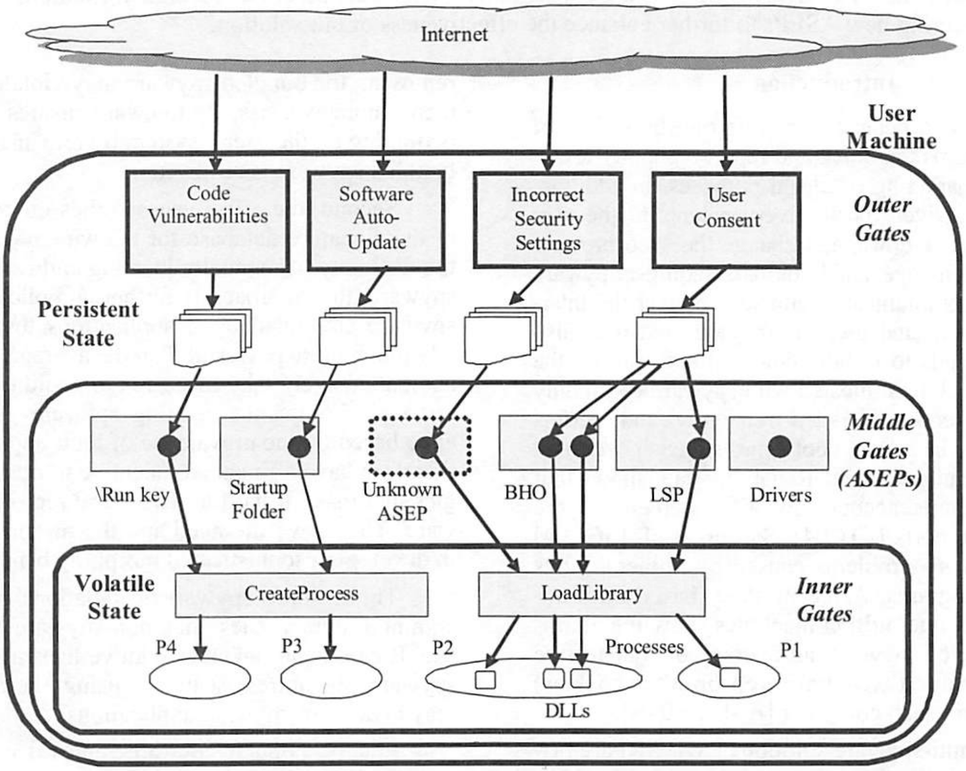


Figure 1: Outer, Middle, and Inner Gates: (1) Outer Gates are the entrance points for program files from the Internet to get on user machines. User Consent includes explicit consent to install, for example, a freeware program, and implicit consent to allow spyware programs bundled with the freeware to get installed as well. Incorrect Security Settings include the "Low" setting for Internet Zone security, incorrect entries in the Trusted Sites list, and incorrect entries in the Trusted Publishers list, which would allow drive-by downloads; (2) Middle Gates are the ASEPs that allow programs to survive reboots and maximize their chance of running all the time. BHO stands for Browser Helper Object. LSP stands for Layered Service Provider; and (3) Inner Gates control the instantiation of program files into active running program instances. They include CreateProcess, LoadLibrary, and other program execution mechanisms, and can be used to block any potentially harmful programs if they are not properly signed or on the known-good list.

Aware and SpyBot-S&D) to remove existing spyware. After Step (1), the Gatekeeper infrastructure is put in place to provide a spyware management framework.

In Step (2), we continuously monitor all ASEPs by recording, alerting, and blocking potentially undesirable ASEP hooking operations. It is essential that the signature database includes user-friendly descriptions of known-good [G03, NSRL, PP] and known-bad ASEP hooks to enable presentation of actionable information to the user.

If the user decides to install a freeware application after assessing the risks of bundled spyware programs as specified in the EULA, bundle tracing in Step (3) captures all components installed by the freeware and display them in Gatekeeper as a group with a user-friendly name enabling the user to manage and remove them as a unit.

In Step (4), we monitor the performance and reliability of the system since the bundle installation and associate any problems with the responsible component(s). These "credit reports" provide the user with a "price tag" for the freeware functionality, enabling the user to make value/cost judgments about the freeware.

Finally, our solution's effectiveness is directly related to completeness of the ASEP list. In Step (5), we discover new ASEPs of OS and popular frequently-run software by either analyzing indirection patterns in file and Registry traces or troubleshooting infected machines. In this paper, we will cover (2), (3), and (5) in the next three sections.

ASEP Management

ASEP Categorization

On Windows platforms, most of the ASEPs reside in the Registry. Only a few of them reside in the file system. We have found it useful to classify ASEPs into the following categories:

1) **ASEPs that start new processes**: for example, the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run Registry key and the %USERPROFILE%\Start Menu\Programs\Startup file folder are well-known ASEPs for auto-starting additional processes.
2) **ASEPs that hook system processes**: for example, HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify allows a DLL to be loaded into WinLogon.exe.
3) **ASEPs that load drivers**: for example, HKLM\System\CurrentControlSet\Control\Class\{4D36E96B-E325-11CE-BFC1-08002BE10318}\UpperFilters allows loading of a keylogger driver; HKLM\System\CurrentControlSet\Services allows loading of general drivers.
4) **ASEPs that hook multiple processes**: for example, Winsock allows a Layered Service Provider (LSP) DLL or a Name Space Provider (NSP) DLL to be loaded into every process that uses Winsock sockets; HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_Dlls allows a DLL to be loaded into every process that links with *User32.dll.*

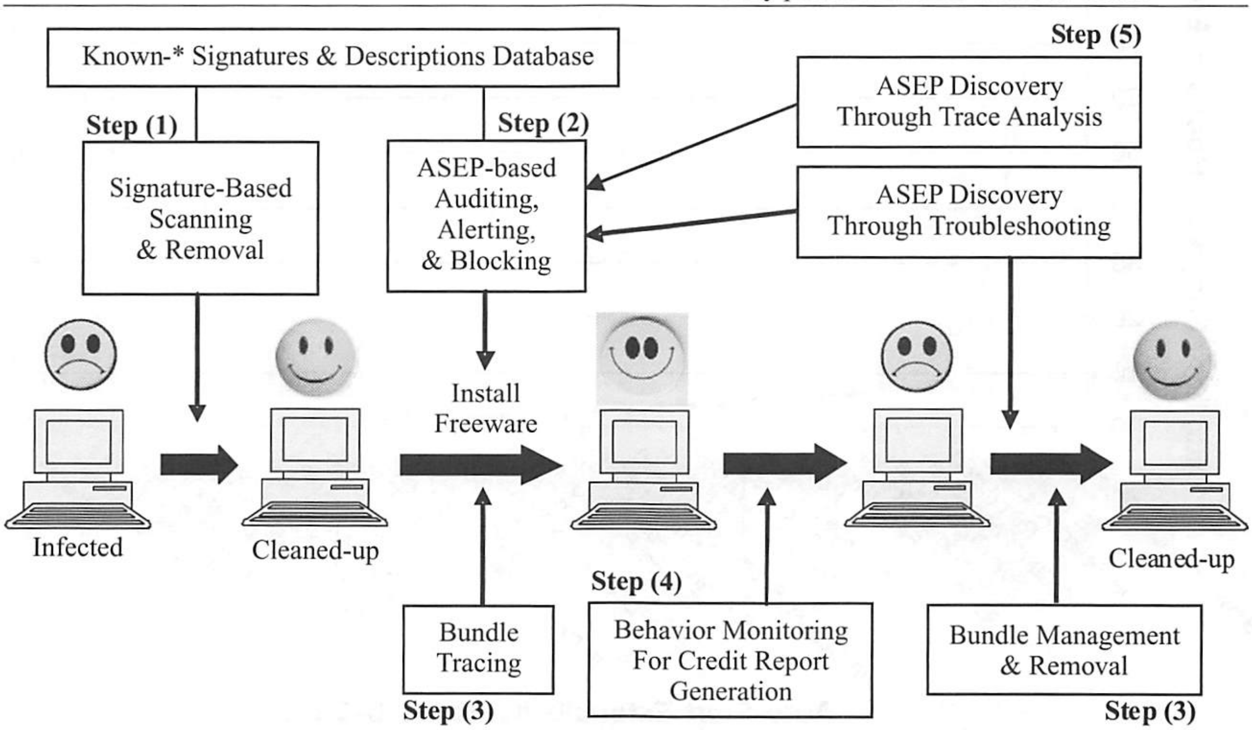


Figure 2: The Spyware Management "Life Cycle" and Problem Decomposition: see descriptions in the Problem Formulation and Decomposition section.

5) **Application-specific ASEPs**: for example, HKLM\SOFTWARE\Microsoft\Internet Explorer\Toolbar allows a toolbar to be loaded into the IE browser; HKCR\PROTOCOLS\Name-Space Handler and HKCR\PROTOCOLS\Filter allow other kinds of DLLs to be loaded by IE; HKLM\SOFTWARE\Microsoft\Internet Explorer\Search\SearchAssistant and CustomizeSearch take URLs as input and control which search pages will be loaded.

ASEP Hooking Statistics

Figure 3 shows the number of spyware hooks to each of the 34 ASEPs hooked by at least one of the 120 spyware programs in our Spyware Zoo. Browser Helper Objects (BHOs), HKLM "Run" key, and IE "Toolbar" are the three most popular ASEPs. Figure 4 shows that most of the individual spyware programs hook only three or less ASEPs, but some hook as many as 13 or 17. When spyware and freeware programs are bundled together in a single installation, it is not uncommon to see that a single bundle hooks 10 or more ASEPs, which would usually cause significant performance degradation. (Note that a freeware program may not have any ASEP hook if it is to be manually launched by the user as needed, but spyware programs always have ASEP hooks.)

ASEP Monitoring and Alerting

ASEP monitoring watches all known ASEPs for any of the following three types of changes: (1) adding a new ASEP hook; (2) modifying an existing ASEP hook; and (3) modifying the executable file pointed to by an existing ASEP hook.

Each of the above changes generates a new event log entry that contains the ASEP pathname, the ASEP hook name, the executable file pathname or URL, and the timestamp of the hooking operation. Optionally, a notification can be displayed to the user or forwarded to an enterprise management system for processing. Notifications for ASEP programs signed by trusted publishers can be optionally suppressed to reduce false positives.

Figure 5 shows a screenshot of a user notification alert. During the installation of a freeware screensaver, the user is notified of five new ASEP hooks. The "Screen Saver" hook alert is obviously expected. Searching the Signatures and Descriptions Database with the information from the other four alerts (by clicking on the alerts) reveals that they belong to "*eXact Search Bar*" and "*Bargain Buddy*." Based on the information provided for these two pieces of software and the benefit provided by the screensaver, the user can then make informed decision about whether to keep this bundle.

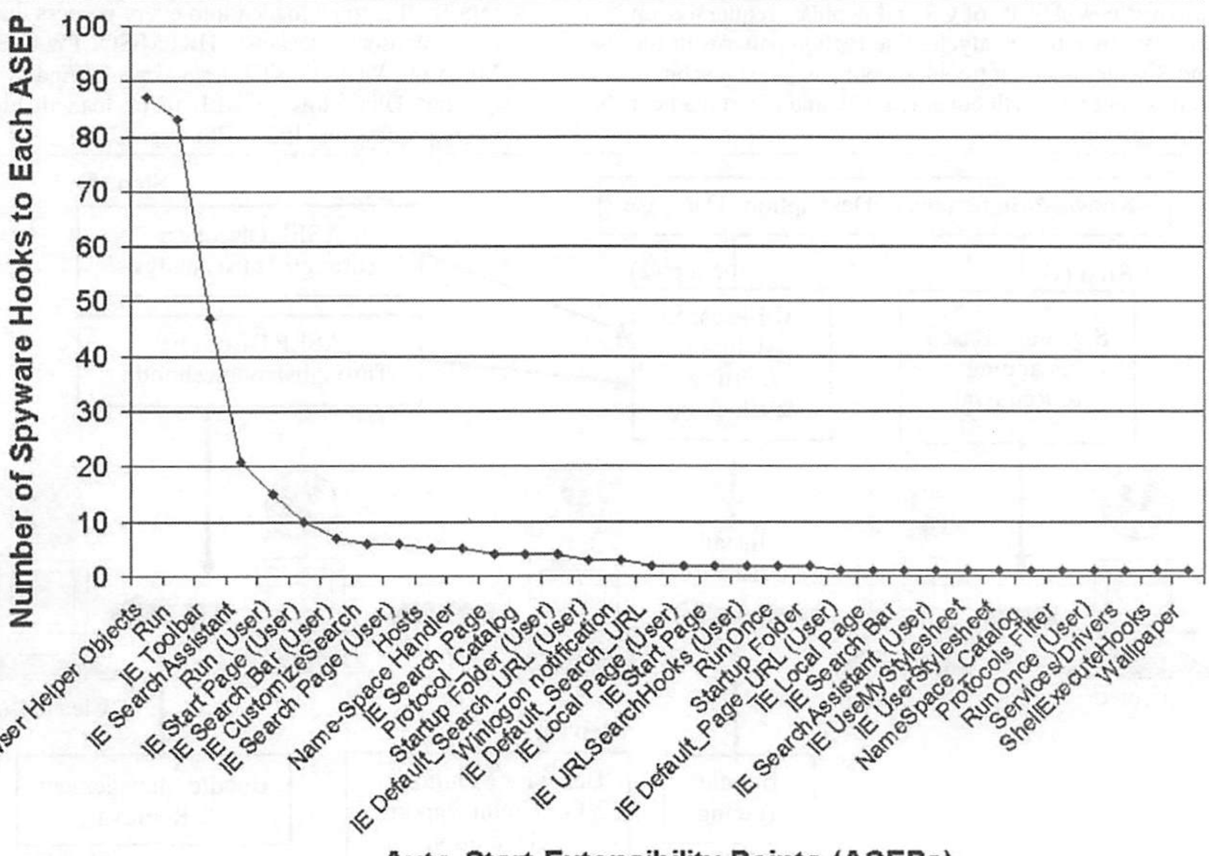


Figure 3: Distribution of spyware ASEP hooks: 120 spyware programs with 334 hooks to 34 ASEPs; ASEPs are sorted by popularity.

Bundle Management

The term 'bundle' represents a set of applications and extensions added to a user's system as part of a single installation process. If a component of a bundle installs additional applications or components at a later time, these are also added to the installer's bundle. A bundle is intended to match an end user's ideal management unit for installing, disabling, and removing software on their system.

Bundle Tracing

Although multiple ASEP alerts appearing during a single installation typically indicate that the ASEPs belong to the same bundle, this time-based grouping is not robust against concurrent installations. For example, Figure 6 illustrates two concurrent installations of the *DivX* bundle (with two ASEP hooks) and the *Desktop Destroyer (DD)* bundle (with five ASEP hooks). Time-based grouping would incorrectly group all seven ASEP hooks in a single bundle.

Gatekeeper uses a bundle tracing technique built on top of the always-on Strider Registry and file tracing [WVD+03, DRD+04]. ASEP hooks created by processes belonging to the same process tree are assigned to the same bundle. If any *Add/Remove Programs (ARP)* entries are created by any process in the tree, the concatenation of their ARP Display Names is used to label the bundle. Referring to Figure 6, the upper process tree defines the *DivX* bundle with two ARP names, and the lower tree defines the *DD* bundle with three ARP names.

Any spyware that does not provide an ARP entry for removal will show up as a bundle with no name. For example, the *ClientMan* software creates one ASEP hook silently at installation time with no accompanying ARP entry. Since installations without ARP entries are uncommon, this installation will be flagged as potentially unwanted.

We have observed that some spyware may initially install partially, and delay the full installation until a later time to make it more difficult for the users to identify which Web site is actually responsible for installing the software. For example, after the partial installation with one ASEP hook, *ClientMan* would non-deterministically select a later time after several reboots to finish its installation with seven additional ASEP hooks.

Gatekeeper bundle tracing captures such devious behavior as follows. First, it performs URL tracing to link each Web-based bundle installation with its source URL. Although IE browser history already records the URL and timestamp for every Web site visited, it is a global history for all instances of IE and is garbage collected after a few weeks. We have implemented a Browser Helper Object to record the process ID of the IE instance that navigated to each URL so that the URL trace can be correlated with the ASEP hooking trace. Second, to handle latent installations, bundle tracing keeps track of all the files created by each bundle. If any of the files are later instantiated to create more ASEP hooks, these additional hooks are added to the original bundle.

Extensibility Point Add/Remove Programs (EP-ARP)

Figure 7 shows Gatekeeper displaying bundle information through a new "*Manage Auto-Start Programs*" button in the Control Panel ARP interface (called it *EP-ARP*). It scans all ASEPs and displays all current hooks by bundles. Users can also choose to

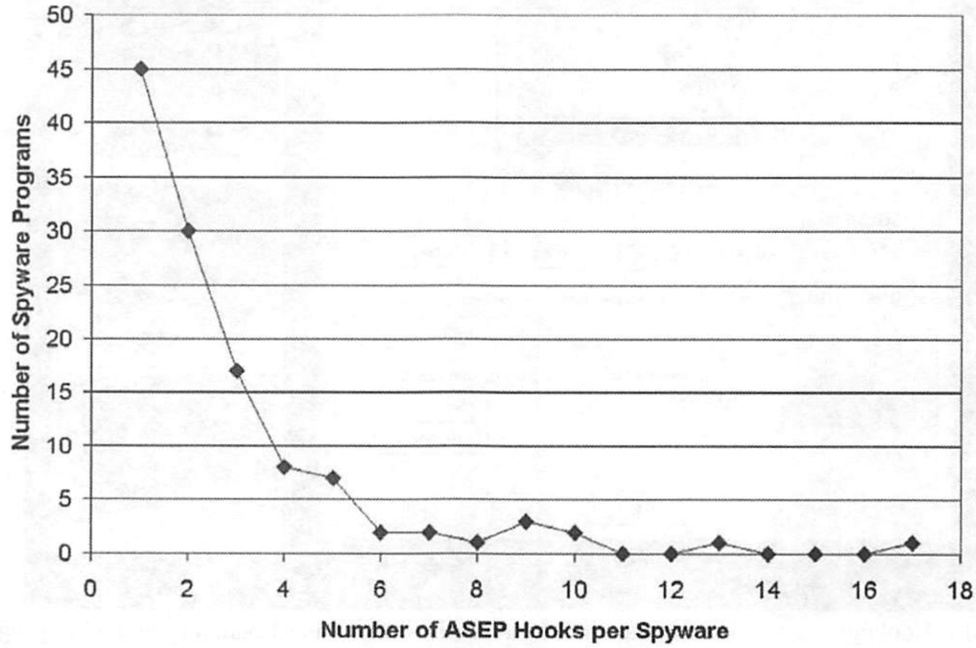


Figure 4: Number of ASEP hooks used by each spyware.

sort the ASEP hooks by the timestamps obtained from the event log in order to highlight newly installed ASEP programs. This is particularly useful when a user invokes EP-ARP immediately after she observes a problem to identify the potential problematic program.

The EP-ARP display also provides three options for bundle removal/disabling. For example, the bundle name clearly shows that "*eXact Search Bar*" and "*Bargain Buddy*" have been installed as part of the *DD bundle*. If the user wants to remove *DD*, she can click the "*Disable Bundle*" button and reboot the machine. This removes all five ASEP hooks, stopping the three bundled programs from automatically starting, despite their files remaining on the machine.

Alternatively, the user can look for the three ARP names in the regular ARP page and invoke their respective removal programs there. Since it is not uncommon for spyware to provide unreliable ARP removal programs, the user can double-check EP-ARP to make sure that none of the ASEP hooks gets left over after ARP removals. Gatekeeper also integrates with System Restore [SR01], as shown at the bottom of Figure 7. If both removal options fail, the user can click on the "*Restore*" button to roll back machine configuration to a System Restore checkpoint taken before the bundles were installed.

ASEP Discovery

In addition to well-known ASEPs and documented ASEPs, we discover new ASEPs through another two channels. The first channel involves troubleshooting machines with actual infections that cannot be cleaned up by Gatekeeper because of spyware using unknown ASEPs. We provide two tools for this purpose: the Strider Troubleshooter [WVD+03] and the automatic *AskStrider* scanner [WRV+04]. The second channel involves analyzing Registry and file traces collected from any machine to discover new ASEPs that can potentially be hooked by future spyware. Once new ASEPs are discovered, they are added to the Gatekeeper list to increase its coverage. The same ASEP discovery procedure can also be used by

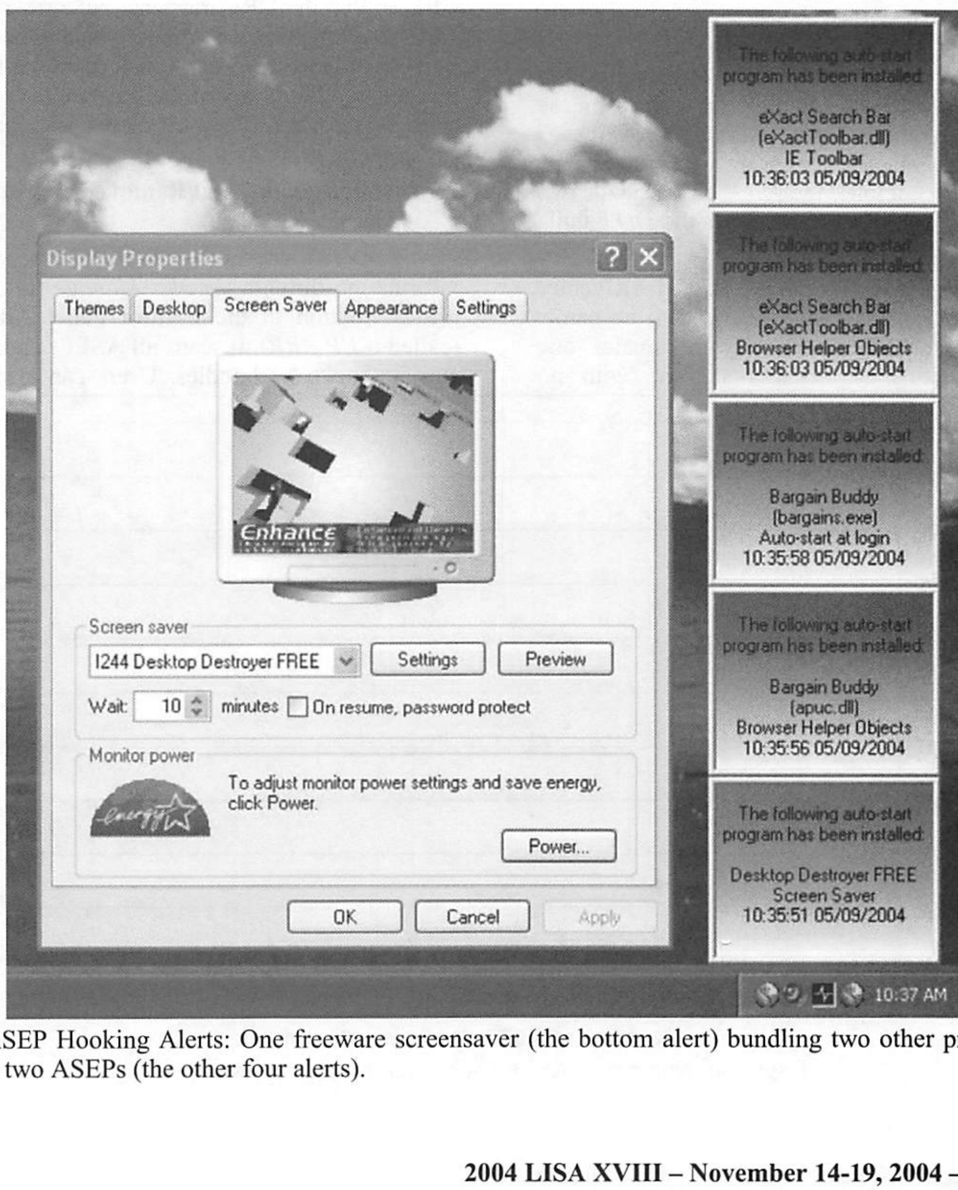


Figure 5: ASEP Hooking Alerts: One freeware screensaver (the bottom alert) bundling two other programs, each hooking two ASEPs (the other four alerts).

system administrators to discover ASEPs in third-party or in-house applications that do not come with a list of specified ASEPs.

ASEP Discovery Through AskStrider

The *AskStrider* scanner is an enhanced Windows Task Manager. In addition to displaying the list of running processes, *AskStrider* displays the list of modules loaded by each process and the list of drivers loaded by the system. More importantly, *AskStrider* gathers context information from the local machine to help users analyze this large amount of information to identify the most interesting pieces. Such context information includes the System Restore file change log, meta-data for patch installations, and driver-device associations [WRV+04].

Figure 8 shows two sample screenshots of *AskStrider*. The upper pane displays the list of processes sorted by the approximate last-update timestamps of their files, according to System Restore. Files that were updated within the past week are highlighted. The lower pane displays the list of modules loaded by the selected process in the upper pane, with the same time-sorting and highlighting. Additionally, if a file came from a patch, the patch ID is displayed as an indication that the file is much less likely to have come from a spyware installation.

Also illustrated in Figure 8 is an example of how *AskStrider* was used to discover a new ASEP. Figure 8 (a) shows that, after the installation of *SpeedBit,* a new process *DAP.exe* was started and the browser process *iexplore.exe* was loading four newly updated DLL files from the same installation. After we disabled all new ASEP hooks from Gatekeeper EP-ARP and rebooted the machine, *iexplore.exe* was still loading two new DLLs as shown in Figure 8 (b). Searching the Registry using the filename *DAPIE.dll* revealed that SpeedBit

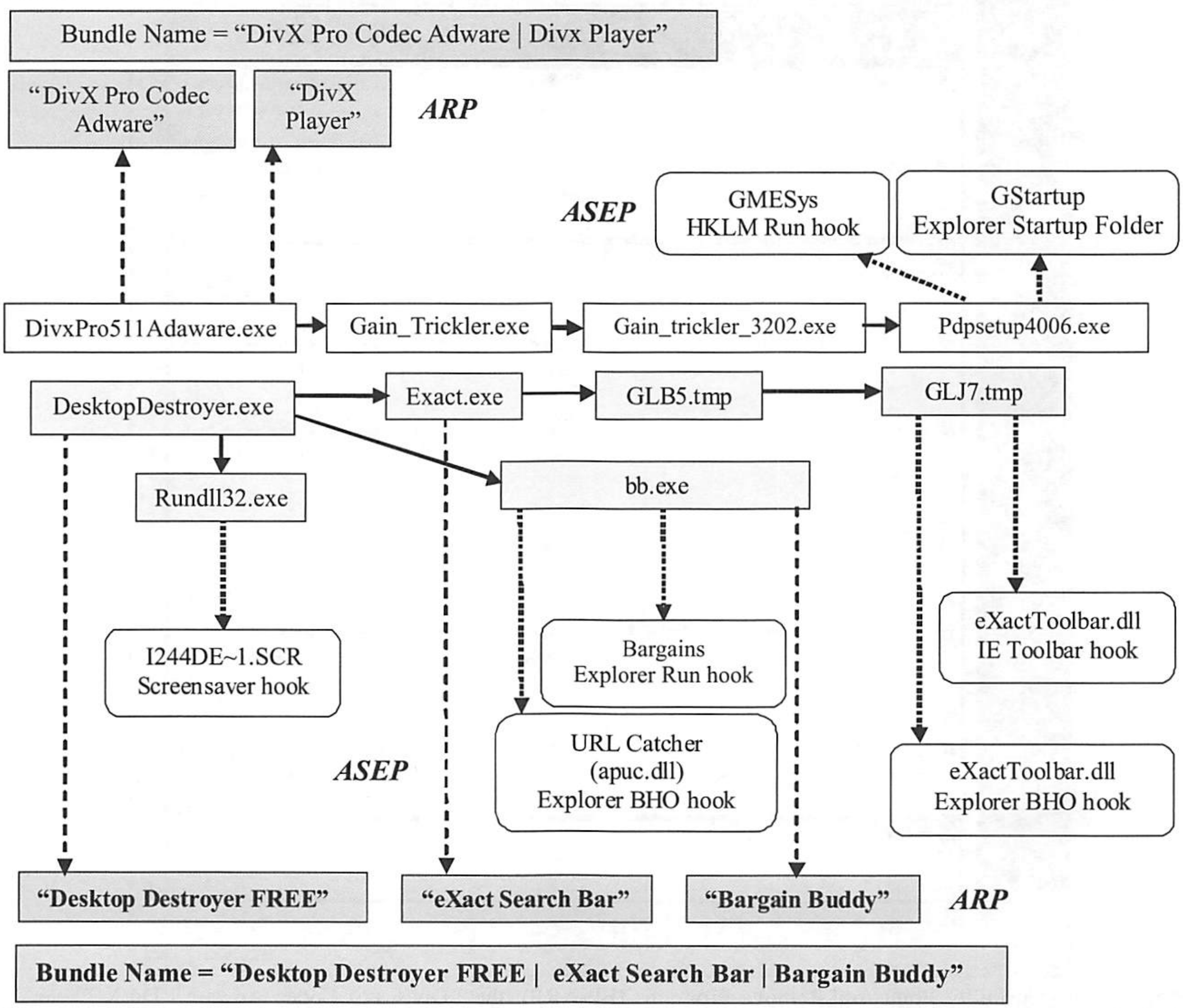


Figure 6: DivX and Desktop Destroyer Bundle Tracing: solid arrows represent creations of child processes; dashed arrows represent creations of ARP entries; dotted arrows represent creations of ASEP hooks. Each process tree defines the scope of the bundle, named by concatenation of ARP friendly names.

was hooking an additional ASEP under HKCR\PROTOCOLS\Name-Space Handler, which has since been added to the ASEP list monitored by Gatekeeper.

ASEP Discovery Through Strider Troubleshooter

The strength of *AskStrider* is that the scanning is completely automatic and typically takes less than a minute to run. The weakness is that it only captures running processes and loaded modules at the time of its scan. If a spyware program gets instantiated through an unknown ASEP and exits before *AskStrider* is invoked, *AskStrider* may not be able to capture any information revealing the unknown ASEP.

The Strider Troubleshooter [WVD+03] can capture such behavior in an "auto-start trace" that records every single file and Registry read/write during the auto-start process. This tool asks the user of an infected machine to select a System Restore checkpoint (of files and Registry) that was taken before the infection. By comparing that checkpointed state with the current infected state, the tool calculates a diff set that contains all changes made by the spyware installation. Then it intersects the diff set with the auto-start trace to produce a report that contains all ASEP hooks made by the spyware installation and accessed during auto-start.

For example, in the case of *Praize Desktop*, HKCU\Control Panel\Desktop\Wallpaper was a previously unknown ASEP that allows running an HTML file as a desktop picture. It did not show up in *AskStrider*, but it showed up in the Strider Troubleshooter report as a newly discovered ASEP.

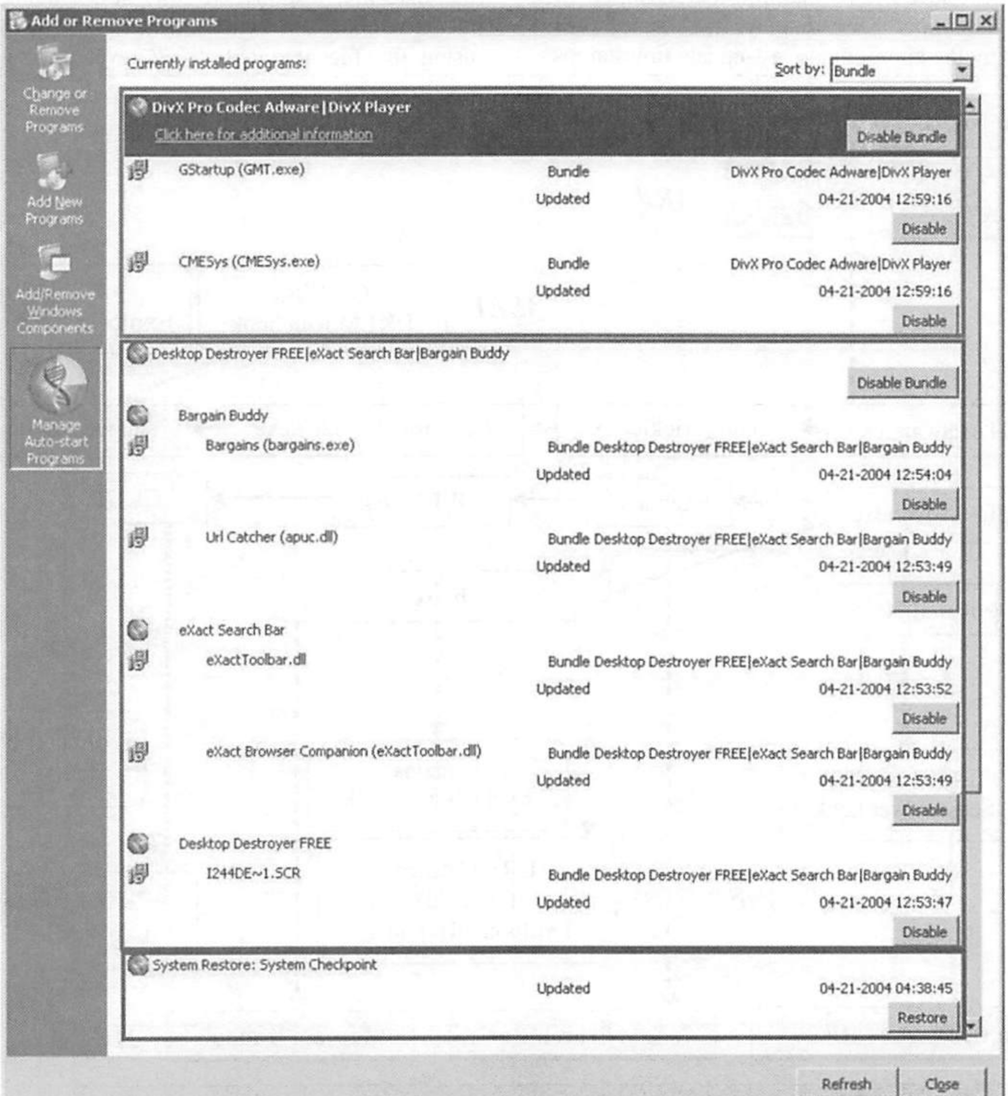
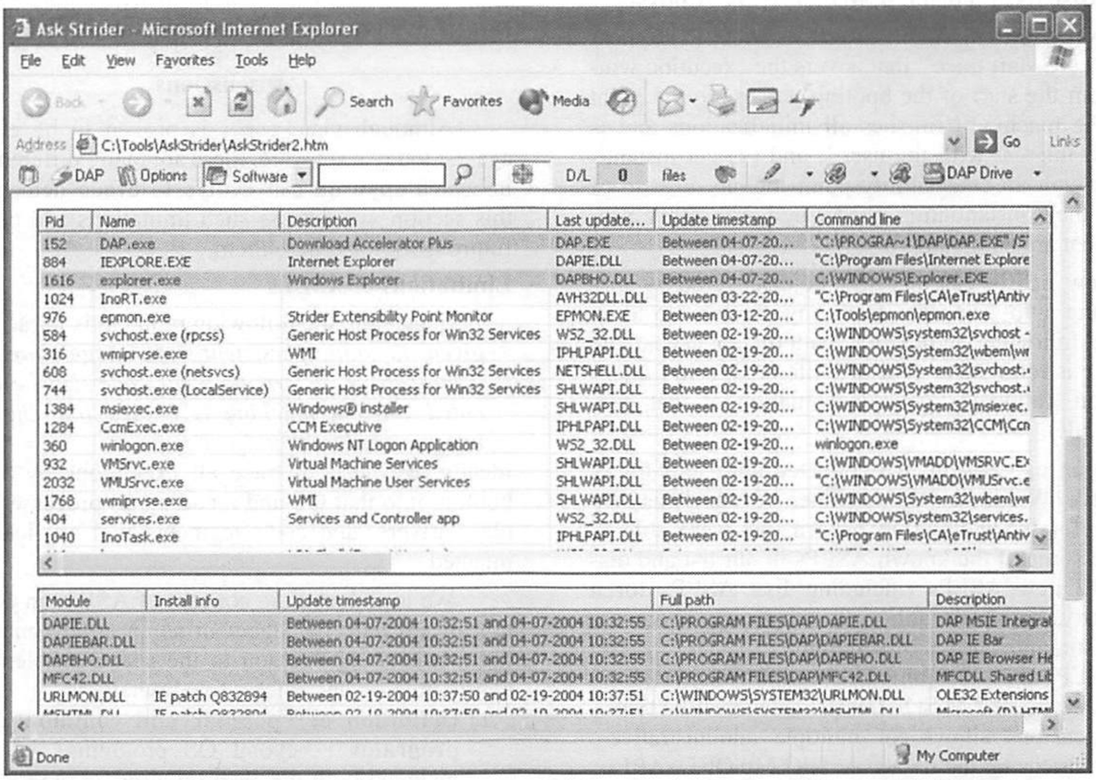
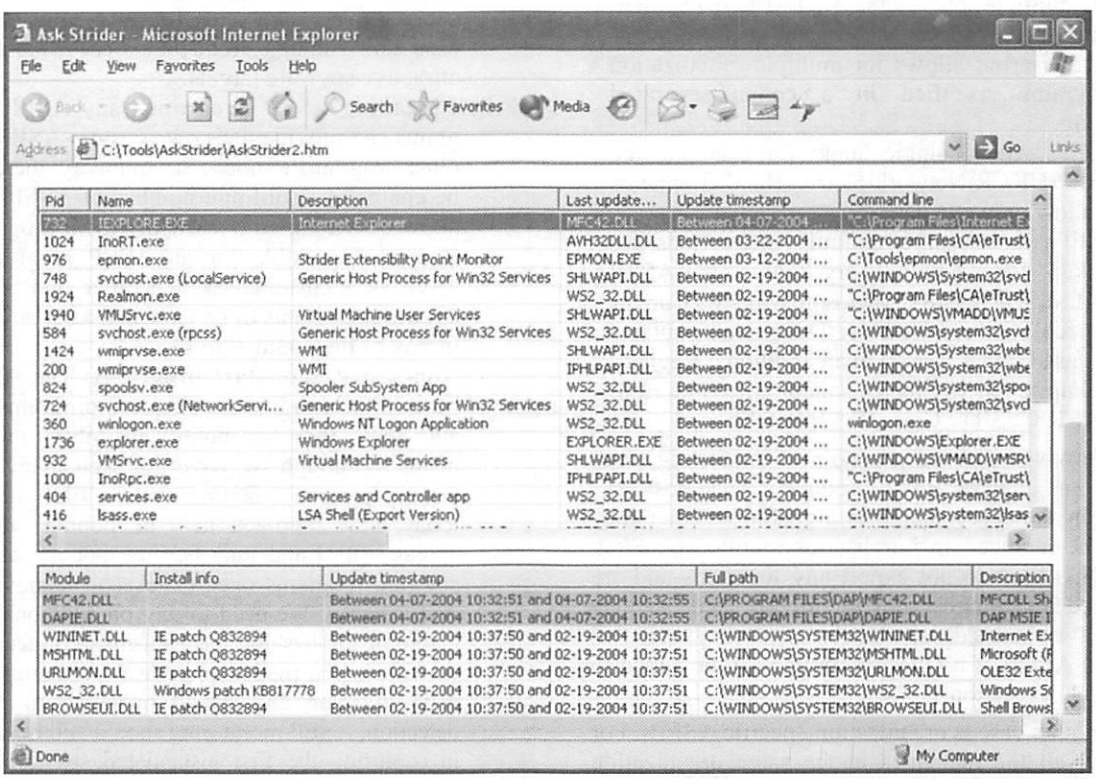


Figure 7: Extensibility Point-Add/Remove Programs (EP-ARP): the "DivX Pro Codec Adware | DivX Player" bundle includes two ASEP hooks GMT.exe and CMESys.exe that came from Gator. The "Desktop Destroyer FREE | eXact Search Bar | Bargain Buddy" bundle includes five ASEP hooks. Clicking on the "Restore" button at the bottom can roll back the system and remove the two bundles.



(a) After Installing SpeedBit: a new process DAP.exe and four new DLLs are highlighted.



(b) After Disabling All New ASEP Hooks from Gatekeeper and Rebooting the Machine: two new DLLs are still loaded through a previously unknown ASEP.

Figure 8: *AskStrider* for ASEP discovery.

ASEP Discovery Through Strider Trace Analysis

By definition, ASEP programs must (1) appear in the "auto-start trace" that covers the execution window from the start of the booting process to the point when the machine "finishes all initializations and is ready to interact with the user"; and (2) get instantiated through an extensibility point lookup, instead of having their instantiation hard-wired into other programs that are auto-started.

New ASEPs can therefore be discovered by analyzing the auto-start trace from any machine to identify the following indirection pattern: an executable filename is returned as part of a file or Registry query operation, followed by an instantiation of that executable file.

In an experiment, we collected auto-start traces from five Windows XP machines for analysis. By looking for the indirection pattern, we were able to validate some of the known ASEPs in our list and discover 17 new ASEPs (including five ASEPs for a third-party, auto-start anti-virus program). There are three distinctive classes of patterns:

1) ASEPs that accommodate multiple hooks: for example, HKLM\SOFTWARE\Microsoft\InetStp\Extensions allows for multiple administrative extensions for the IIS server; HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider allows for multiple providers; HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\Userinit allows for multiple initialization programs specified in a comma-separated string.
2) ASEPs with a single hook: for example, the ASEP HKCR\Network\SharingHandler appears to allow only one handler.
3) ASEPs that require multiple indirections for lookup: for example, every hook to the ASEP HKLM\SOFTWARE\Microsoft\Windows\Current Version\ShellServiceObjectDelayLoad contains a Class ID that is used in an additional Registry lookup to retrieve the executable filename from HKCR\CLSID\<Class ID>\InProcServer32.

We have observed a couple of interesting cases where our analysis may produce "false-positive" ASEPs in the sense that it is arguable whether they should be included in our list for monitoring. First, some DLL files do not export any functions and are only used as resource files to provide data; so they may not be considered ASEPs. But they are still potential ASEPs if a *DllMain* routine can be added to cause code execution.

Another case is organization-specific ASEPs. For example, all the machines in the same organization may run an auto-start program deployed by its IT department that exposes its own ASEPs. Obviously, such ASEPs should not be added to the global list; but the system administrators in the organization may want to add them to their local list if they are concerned about these ASEPs being hooked.

Discussions

Although Gatekeeper is proven to be effective against today's spyware, there are many different ways in which spyware can evolve to evade detection. In this section, we discuss such limitations and potential future work to address them.

Limitations of ASEPs

In general, the following problem is intractable:

> *given the static persistent-state image stored on a hard drive, determine what code will be executed when a machine is booted into the OS image stored on that drive.*

Ideally, we need to trace all executions by actually booting into that OS and recoding all processes, modules, drivers, and code segments that are loaded or injected.

We introduced the concept of ASEPs in the context of spyware management as an approximation to the non-existing solution to the above problem. This approximation has at least five limitations:

1) **Definition of "popular and commonly run programs"**: beyond OS programs, we have included only the Web browser for ASEP consideration. Many commercial or freeware applications may have a sufficiently large install base and running frequency that make them attractive spyware targets.
2) **Cascading ASEP programs**: any ASEP programs can provide their own custom ASEPs for other programs to hook. So, in theory, there can be chains of an infinite number of ASEPs that allow cascading auto-starts of ASEP programs. It is also possible for an ASEP program to serve as a custom task scheduler that allows spyware programs to be launched after any definition of the "auto-start phase."
3) **Non-ASEP auto-start programs**: The ASEP-based approach does not capture programs that auto-start through non-extensibility mechanisms. Although we have not seen many spyware programs infecting system files directly today, that approach may be popular among Trojans [I04] and may become popular among spyware programs once Gatekeeper exposes all ASEP hooks. We need to rely on additional signature or file-hashing mechanisms to protect system files. If a malicious spyware program uses code injection and thread hijacking to evade detection, ASEP monitoring should still be useful in capturing the first instantiated spyware program. In theory, it may also be possible for a program to hide inside an input file and get instantiated when the file is read by an auto-start program by exploiting code vulnerabilities.

4) **ASEP hijacking**: Gatekeeper assumes that the underlying operating system has not been compromised, so the list of ASEPs on a spyware-infected machine is the same as that on a clean machine. It is possible that a malicious spyware program can "hijack" the ASEPs by replacing system files; essentially, the machine can be considered to be running a different operating system in such cases. For example, a Web posting [CAR04] describes a way to modify the binary file of explorer.exe to create arbitrary ASEPs. In our current work, we consider such malicious programs targets of anti-virus programs, not Gatekeeper. In our future work, we plan to rely on digital signatures and file hashes to verify that the underlying operating system is not compromised.
5) **ASEP hook hiding**: Another way for malicious software to defeat ASEP-based scanning is to intercept all file and Registry query operations and remove the software's own ASEP hooks from the query results before they are returned to Gatekeeper. Many RootKits are known to provide such capability [P03]. There have been recent reports that an open-source RootKit is being used to hide spyware programs from anti-spyware tools [HD]. We plan to augment Gatekeeper with an external scanning mechanism, which is required to combat such malicious programs that essentially take over the entire machine once they get started [WVR+04].

Finally, the operating system can be configured to auto-start programs based on generated system events resulting from the insertion of removable media like CDs, hot-pluggable hardware like USB key rings, etc. However, as long as they require explicit user actions to connect the media to the machine and are not automatically started upon reboot, they are not considered ASEPs in this paper.

Bundle Management Challenges

Our bundle tracing technique assumes that the spyware installation programs that we monitor do not try to intentionally confuse or maliciously attack Gatekeeper. One can imagine that a deceptive spyware could hijack ARP and ASEP hook entries of other good software so that they are incorrectly included in the bad bundle. A malicious spyware running with administrator privileges could even disable the bundle tracer or modify the recorded bundle information.

There are two additional challenges that do not involve malicious behavior. First, since our bundle tracer does not track inter-process communications, any bundle installation that involves communications between multiple process trees may appear as multiple, separate bundles. Second, if two toolbars from two different bundles are loaded into the browser at the same time and one of them expands the bundle by hooking an additional ASEP, process tree-based tracking will not provide sufficiently fine-grain information to determine which bundle the new ASEP hook should belong to.

Limitations of *AskStrider*

AskStrider extracts the last-update timestamps of files from the System Restore file change log and is therefore subject to its limitations. First, System Restore only monitors files with certain filename extensions [SRM]; spyware programs with extensions outside the monitored set will not be captured by *AskStrider's* highlighting of recent changes because their updates will not be captured in the file change log.

Second, System Restore excludes certain temporary folders from monitoring. As a result, file updates inside those folders will not be captured in the change log. Third, a malicious spyware program may delete or corrupt the file change log or hide its processes and modules from the *AskStrider* scan.

Finally, a main feature of *AskStrider* is that it highlights recent changes to aid troubleshooting. We have found that such filtering mechanisms are essential for reducing the complexity in many systems management problems. An obvious limitation of this approach is that it will not work well if a user invokes *AskStrider* long after a spyware installation.

Other Web Browsers and Non-Windows Platforms

In this section, we study the ASEPs and spyware-related issues in other Web browsers and operating systems.

Other Web Browsers

As shown in the Outer Gates in Figure 1, code vulnerabilities can be used as an infection vector for spyware through "drive-by downloads." The problem is not unique to the IE browser; other browsers also have known vulnerabilities, such as Mozilla [KVM] and Netscape [HN00]. Exploits of vulnerabilities in the Mozilla and Firefox Web browsers have been widely publicized in news articles [M04, MO04, BZ]. Also, Secunia Advisories [SEC] often describe vulnerabilities that affect the Opera and Mozilla Web browsers.

Other browsers also expose extension mechanisms similar to Windows ActiveX as another infection vector for spyware through plug-in installations with explicit or implicit user consent. Those affecting Mozilla have used a *.xpi* file which is essentially a *.zip* file containing a JavaScript installer and the files/directories to install [CNPM]. An example of this is the Flingstone XPI extension at http://www2.flingstone.com/cab/sbc_netscape.xpi which contains *install.js* and *sbc_netscape.exe*. When installed through Mozilla Firefox, Flingstone adds several ASEP hooks to the BHO and HKLM Run key. This infects the Windows OS in addition to IE although it does not appear to infect Firefox itself [DSM04]. The bundle includes the well-known software bridge.dll [FB04]. We also found a Flingstone-clone *ist_netscape.xpi* containing

install.js and *istinstall_netscape.exe* and exhibiting essentially the same behavior.

Just like IE, other Web browsers expose ASEPs that can potentially be hooked by spyware. For example, Mozilla Firefox has a file system-based ASEP at C:\Program Files\Mozilla Firefox\plugins; all plug-in DLL files placed in that directory are automatically loaded by Mozilla. It also scans a Registry-based ASEP at HKLM\SOFTWARE\MozillaPlugins to locate plug-ins that register with the browser through PLIDs [PLUG].

The homepage and search page related ASEPs of non-IE browsers are generally stored in application specific preference files rather than the Registry. For example, there are two user preference files in the profile directory of Netscape/Mozilla: *prefs.js* that contains automatically generated default preferences, and *user.js* which contains options that override settings in *prefs.js*. Spyware can hijack the home page and the default search page of these browsers by altering the value of user_pref("browser.startup.homepage," "<home page>") and user_pref("browser.search.defaultengine," "<search page>") in *prefs.js* [NCPI]. For example, the Lop.com software has been known to hijack Netscape/Mozilla home page [LOP]. In general there appears to be less spyware targeting non-IE browsers, presumably because their smaller install base is less attractive to spyware developers.

In some cases, the search and download functionalities of the browser software itself may raise similar privacy concerns. It was reported [NNB02] that, while data on searches conducted from IE's search pane was sent directly to the designated search site and was not intercepted by Microsoft, searches performed by using Netscape Navigator's Search button were intercepted by Netscape and tagged with information that can potentially identify individual machines. The term "*File Download Spyware*" [FDS00] refers to those file downloaders that by default track user's entire file download history tagged with a unique ID, the machine's IP address, or even the user's personal email address.

Non-Windows Platforms

ASEPs on UNIX operating systems such as Linux, AIX, and Solaris can be roughly classified into four categories:

1) **The *inittab* and *rc* files**: The file /etc/inittab instructs the *init* process what to do when the system is up and initializing. It typically asks *init* to allow user logons (*gettys*) and start all the processes in the directories specified by the /etc/rc.d/rc file and other *rc* files such as /etc/rc.d/rc.local, which is a common place for the root user to customize the system, including loading additional daemons.
2) **The *crontab* tool**: The *cron* daemon is started from either the *rc* or the *rc.local* file, and provides task scheduling service to run other processes at a specific time or periodically. Every minute, cron searches /var/spool/cron for entries that match users in the /etc/passwd file and also searches /etc/crontab for system entries (note that any modification to this file requires root privileges.) It then executes any commands that are scheduled to run.
3) **Configuration profiles for user environment** (such as *.bash* for bash shell, *.xinitrc* or *.Xdefaults* for X environment, and other profiles in /etc/) are potential ASEPs. Users are typically unaware of what is loaded when they log on, or start an X window session. A simple script file that contains the command

   ```
   script -fq /tmp/.syslog
   ```

   can be used to hook an ASEP to record the terminal activities of the whole system or a specific user account, depending on the ASEP location. The recording is usually stored in a hidden file (i.e., a filename that begins with a ".") under the global-writable /tmp directory
4) **Loadable Kernel Modules (LKMs)** are units of object code that can be dynamically loaded into the kernel to provide new functionalities. By default most LKM object files are placed in the directory /lib/modules. However, some customized LKM files can reside anywhere on the system [LKMP]. The programs *insmod* and *rmmod* are responsible for inserting and removing LKMs, respectively.

Our preliminary investigation shows that spyware is not a substantial threat to the current Unix/Linux world. Perhaps this is because Unix/Linux has a much smaller install base than Windows in the consumer desktop market, which makes it less attractive to spyware writers. Another reason might be that most Unix/Linux users do not run as administrators; many, if not most, of the spyware programs require administrator privileges to install and run. Finally, Unix/Linux users who do run as administrators are advanced users who are unlikely to fall into the trap of installing spyware.

Related Work

Earlier versions of commercial anti-spyware programs focused on the signature-based, on-demand scanning approach. The latest Ad-Aware Ad-Watch real-time monitor [AP] and Spybot-S&D TeaTimer [ST] provide real-time monitoring similar to Gatekeeper ASEP monitoring. But they do not seem to include centralized auditing and bundle tracing, and the context information that they provide to the users is limited, making them less effective as a management solution as compared to Gatekeeper. On the other hand, they put more emphasis on blocking and protection. The Autoruns tool [AR04] and the Windows XP SP2 IE Add-on Manager both cover only a subset of ASEPs known to be hooked by spyware.

An alternative approach to combating spyware programs is to cut off their communications with remote servers so that collected personal information will not be sent out. One way to achieve this is to use the *Hosts* file to map all blacklisted host names to the local loopback address [BUP]. This approach essentially applies known-bad signatures to the host names and similarly lacks the context information for proper spyware management. Moreover, it addresses only the privacy issue, not the reliability and performance issues.

Saroiu, et al. [SGL04] presented a measurement study of four widespread spyware programs in a university environment by analyzing a week-long trace of network activity. Their results showed that the spyware problem is of large scope. They also described a specific vulnerability in actual spyware programs to demonstrate that the potential for spyware to introduce substantial security problems is real.

Summary

In this paper, we have modeled the spyware management problem as an ASEP tracking and bundling problem. We have described the Gatekeeper solution that provides visibility into important system changes and answers the following critical questions for every potential spyware program:

1) **"Where did it come from?"** Our URL source tracing identifies the Web site from which the program was downloaded; bundle tracing identifies the freeware that bundled the spyware.
2) **"When was it installed, where was it installed, and what was installed?"** Our ASEP monitoring detects and records the installation events, and context lookup determines which file is installed where.
3) **"How does it get instantiated?"** The ASEP to which the program is hooking determines how it will get auto-started.
4) **"How do I disable/remove it?"** Our extended Add/Remove Programs user interface exposes each ASEP hook and allows simple disabling; alternatively, the ARP entries that are bundled with the ASEP hooks can be used for removal.

With these capabilities, the Gatekeeper tool in its current form is useful for technical users and system administrators to gain back control of their machines and to effectively manage spyware. But there remains one critical piece to the puzzle to make the tool useful and make the presentation actionable by average users:

5) **"What does it do?"** This will require detailed experiments and analysis of the program and matching the program's behavior against a list of objective criteria. It will then allow the user to make an informed decision about whether to remove the program, based on the trade-off between the benefit and the potential privacy/security/reliability/performance concerns of all the bundled programs.

Author Information

Yi-Min Wang manages the Systems Management Research Group and leads the Strider project at Microsoft Research, Redmond. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include systems and security management, fault tolerance, home networking, and distributed systems.

Roussi Roussev is currently a Ph.D. student at Florida Institute of Technology. He was an intern at Microsoft Research in 2003 and 2004. His research interests include security, systems management and software verification.

Chad Verbowski has been a Development Lead at Microsoft for the past six years working on multiple Windows OS components and Systems Management software. Previously Chad worked for several years building and deploying management systems in real world environments. He is currently with Microsoft Research. Chad can be reached at chadv@microsoft.com .

Aaron Johnson has contracted with the Systems Management Research Group at Microsoft Research. He received a Bachelors degree in Computer Information Systems in 1990 from DeVry Institute of Technology in Phoenix. He has nine years of experience troubleshooting Windows hardware and software problems.

Ming-Wei Wu is a software engineer at Institute for Information Industry and the chair of TWING (TaiWan Internet Next Generation) at Taiwan Network Information Center. He received his MS degree from National Chiao Tung University in 2003 and has been a Ph.D. candidate in Electrical Engineering at National Taiwan University since 2004. His research interests include network security, P2P networking and fault tolerance.

Yennun Huang received his MS and Ph.D. degrees from University of Maryland. He worked for AT&T/Lucent Bell Labs for 12 years and was the Department Head of the AT&T Dependable Computing Research Department before he joined a startup as VP of Engineering in 2001. His research interests include dependable distributed computing, mobile infrastructure and applications, and middleware.

Sy-Yen Kuo has been with National Taiwan University since 1991 and was Head of Department of Electrical Engineering from 2001 to 2004. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1987, was a faculty member in the Department of Electrical and Computer Engineering at the University of Arizona from 1988 to 1991, and was the Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan from 1995 to 1998. His current research interests include software reliability engineering, mobile computing, and dependable systems and networks.

References

[AA] *Ad-Aware*, http://www.lavasoft.de/ms/index.htm .

[AP] *Ad-Aware Plus*, http://www.lavasoftusa.com/software/adawareplus/ .

[AR04] *Autoruns*, http://www.sysinternals.com/ntw2k/freeware/autoruns.shtml .

[BUP] *Blocking Unwanted Parasites with a Hosts File*, http://www.mvps.org/winhelp2002/hosts.htm .

[BZ] *Bugzilla Bug 249004*, http://bugzilla.mozilla.org/show_bug.cgi?id=249004 .

[C04] *Spyware Cures May Cause More Harm Than Good*, http://news.com.com/2100-1032-5153485.html , Feb, 2004.

[CAR04] "Create New Autorun by Patching Explore.exe," *The Online Rootkit Magazine*, http://www.rootkit.com/newsread.php?newsid=118 , 2004.

[CNPM] *Creating New Packages for Mozilla*, http://www.mozilla.org/docs/xul/xulnotes/xulnote_packages.html .

[DRD+04] Dunagan, John, Roussi Roussev, Brad Daniels, Aaron Johson, Chad Verbowski, and Yi-Min Wang, "Towards a Self-Managing Software Patching Process Using Black-Box Persistent-State Manifests," in *Proc. Int. Conf. Autonomic Computing*, May, 2004.

[DSM04] "Discussion of sbc_netscape.xpi from MozillaZine," http://forums.mozillazine.org/viewtopic.php?t=66531 .

[E04A] "EarthLink Finds Rampant Spyware, Trojans," *InfoWorld*, http://www.infoworld.com/article/04/04/15/HNearthspyware_1.html , April 15, 2004.

[E04B] "FTC to Look Closer at Spyware," *Washington Post*, http://www.washingtonpost.com/wp-dyn/articles/A22514-2004Apr18.html, April 19, 2004.

[FB04] *Flingstone Bridge*, http://www.kephyr.com/spywarescanner/library/flingstonebridge/index.phtml .

[FDS00] *The Anatomy of File Download Spyware*, http://grc.com/downloaders.htm .

[FTC04] "Monitoring Software on Your PC: Spyware, Adware, and Other Software," *Workshop Transcript*, Federal Trade Commission, http://www.ftc.gov/bcp/workshops/spyware/transcript.pdf .

[G03] Garfinkel, Simson L., *A Web Service for File Fingerprints: The Goods, the Bads, and the Unknowns*, http://www.simson.net/clips/2003.15_972.FinalPaper.pdf .

[HD] "Nasty New Parasite," *SpywareInfo Newsletter*, June 18, http://www.spywareinfo.com/newsletter/archives/0604/8.php .

[HN00] "Hacker finds hole in Netscape," *Wired*, http://www.wired.com/news/technology/0,1282,38087,00.html .

[I04] *Institution 2004 Remote Admin Tool*, http://www.evileyesoftware.com/ees/content.php?content.43 .

[LKMP] *The Linux Kernel Module Programming Guide*, http://tldp.org/LDP/lkmpg/ .

[KVM] *Known Vulnerabilities in Mozilla*, http://www.mozilla.org/projects/security/known-vulnerabilities.html .

[LOP] *Lop.com ("Live Online Portal") Parasite*, http://www.doxdesk.com/parasite/lop.html .

[M04] "Mozilla Flaw Lets Links Run Arbitrary Programs," *eWeek*, http://www.eweek.com/article2/0,1759,1621451,00.asp , July 8, 2004.

[MO04] "Mozilla to squash security bugs," *CNET News.com*, http://news.com.com/Mozilla+to+squash+security+bugs/2100-1002_3-5286138.html , July 27, 2004.

[NCPI] *Netscape Communicator Preferences Index*, http://developer.netscape.com/docs/manuals/communicator/preferences/ .

[NNB02] *Netscape Navigator Browser Snoops on Web Searches*, http://www.computeruser.com/news/02/03/08/news5.html .

[NSRL] *National Software Reference Library (NSRL) Project Web Site*, http://www.nsrl.nist.gov/ .

[P03] Poulsen, Kevin, "Windows Root Kits a Stealthy Threat," *SecurityFocus*, http://www.securityfocus.com/news/2879, Mar 5, 2003.

[PLUG] *PluginDoc for Mozilla*, http://plugindoc.mozdev.org/notes.html#scan-acroread .

[PP] *Pest Patrol*, http://research.pestpatrol.com .

[SB] *Spybot-S&D*, http://www.safer-networking.org/microsoft.en.html .

[SEC] *Secunia Advisories*, http://secunia.com/advisories/ .

[SGL04] Saroiu, S., S. D. Gribble, and H. M. Levy, "Measurement and Analysis of Spyware Infections in a University Environment," *Proc. of the First USENIX/ACM Symp. on Networked Systems Design and Implementation (NSDI)*, 2004.

[SR01] *Windows XP System Restore*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/windowsxpsystemrestore.asp .

[SRM] *System Restore Monitored File Extensions*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sr/sr/monitored_file_extensions.asp .

[ST] *Spybot-S&D TeaTimer*, http://www.safer-networking.org/en/faq/33.html .

[WRV+04] Wang, Yi-Min, Roussi Roussev, Chad Verbowski, Aaron Johnson, and David Ladd, "AskStrider: What Has Changed on My Machine Lately?," *Microsoft Research Technical Report MSR-TR-2004-03*, Jan, 2004.

[WVD+03] Wang, Yi-Min, Chad Verbowski, John Dunagan, Yu Chen, Helen J. Wang, Chun Yuan, and Zheng Zhang, "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. Usenix Large Installation Systems Administration (LISA) Conference*, pp. 159-171, October 2003.

[WVR+04] Wang, Yi-Min, Binh Vo, Roussi Roussev, Chad Verbowski, and Aaron Johnson, "Strider GhostBuster: Why It's A Bad Idea For Stealth Software To Hide Files," *Microsoft Research Technical Report MSR-TR-2004-71*, July 2004.

A Machine-Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing

Sufatrio – Temasek Laboratories, National University of Singapore
Roland H. C. Yap – School of Computing, National University of Singapore
Liming Zhong – Quantiq International

ABSTRACT

The number of security vulnerabilities discovered in computer systems has increased explosively. Currently, in order to keep track of security alerts, system administrators rely on vulnerability databases such as: CERT Coordination Centre, Securityfocus BugTraq and Sans Vulnerabilities Notes Database. Such databases are designed primarily to be read and understood by humans. Given the speed at which an exploit becomes available once a vulnerability is known, and the frequency of occurrence of such vulnerabilities, manual human intervention is too slow, time-consuming and may not be effective. We propose the design of a new vulnerability database which is oriented to be machine readable and processable rather than human oriented. This allows automated response to a vulnerability alert rather than relying on manual intervention of system administrators. With this approach, many kinds of automatic processing of alerts become feasible. We show the value of such a database by constructing a prototype sample scanner for Unix systems tailored for Linux RedHat and FreeBSD. We envisage that our work can help spur a development of far more effective vulnerability databases to benefit a wide-ranging user community.

Introduction

A worrying trend in the age of the Internet is the increasing incidence of cyber attacks. CERT statistics [1] quotes 114,855 reported incidents (an incident may involve an arbitrary number of sites, even thousands) in the first nine months of 2003 alone. This is a large jump from 21,756 incidents in 2000.

One of the objectives of computer security emergency centers like CERT is to help disseminate vulnerability alerts and relevant advisory notes to the user community in a timely fashion. However, the speed of cyber attacks together with the complexity of administrating computer and network infrastructures today, makes it difficult for many system administrators to cope with such attacks. While automatic tools may be available, there is still a need to routinely inspect any security/vulnerability alerts in order to take the necessary corrective measures.

Current sources of such alerts are designed primarily for human consumption and contain large amounts of information in natural language format. In this paper, we will call such sources, *vulnerability databases*, because they deal with collections of data and not whether they are actually kept in a database form or not. While a human oriented format is useful for disseminating the full details of an alert, it also requires a human in the chain to make use of it. This problem is acknowledged in a CERT document [2]. Given the 5500 vulnerabilities reported in 2002, it is estimated that a system administrator would need 229 days just to digest the information. Furthermore, usually multiple vulnerability databases need to be consulted to fully deal with a vulnerability, i.e., just the CERT entry is not sufficient. Thus, the deck is stacked on the side of the hackers rather than the system administrators.

Clearly, the solution would be to move away from direct human processing towards automatic security alert response processing. This paper proposes an initiative to redesign vulnerability databases to be machine oriented and amenable to automatic processing. In practice, such a database would also need to integrate vulnerabilities disclosed from multiple sources. The dissemination of machine processable alerts allows for automated tools to operate on an alert immediately without requiring humans in the loop. This would cut down the long time interval between release of a vulnerability/advisory note and corrective action being taken. Other automated tools do exist, e.g., Microsoft Windows systems have Software Update Services, however there is little which is general purpose, publicly accessible, and open to public or third party scrutiny and verification. We have developed a proof-of-concept machine oriented database schema using a vulnerability expression language for describing the targets and effects of vulnerabilities. To illustrate the use of this database, we have developed a prototype vulnerability scanning robot which can determine existing and potential vulnerabilities based on the database.

The creation of an effective machine oriented vulnerability database would require the cooperation of many parties such as CERT, BugTraq, vendors, software developers, etc. As such, this paper is not meant to be a standalone definitive solution. Rather the prototype database and scanner is intended to spur the development of machine oriented databases by the parties concerned. We believe that our proof of concept presents the key elements for further development of machine oriented vulnerability databases. The use of a simple vulnerability expression abstraction also simplifies the integration of data from multiple sources.

Motivation and Design Goals

Figure 1 reproduced from the following CERT report [3] describes the vulnerability exploit cycle. The Y-axis represents the number of incidents for a given vulnerability.

The graph illustrates the time lag between the release of a vulnerability/advisory report and the decrease of incidents following corrective measures by users. We argue that current vulnerability databases, such as CERT, Bugtraq, CVE, in their present format are not designed to facilitate a speedy user response because they suffer from the following limitations:

1. Much of the information in these databases, particularly the portions which relate to dealing with a vulnerability, is only in a human readable free-text format. While this may be necessary to convey the full information content, it also means that a human needs to interpret the database entry. This makes it difficult to have any form of automated machine processing of this information. While it is possible to analyse the natural language text, this may introduce more problems due to ambiguities in natural language.
2. Different response centers use different terminologies and conventions in describing one vulnerability, which may confuse the users of the information. For instance, some databases put the affected systems according to vendor's version (e.g., RedHat Linux 8.0). A different vulnerability might refer to the Linux kernel version instead.
3. There is a conflicting and fluctuating standard among response centers which actively promote their own methods and standards, thus causing frequent shifting and switching among different proposed standards.

Our philosophy is that vulnerabilities should be expressible in an explicit form in terms of data (or a description) rather than an implicit form like code to process a vulnerability. Hence the data can be stored in a database (or any data description language, i.e., XML). Our database is designed with the following goals:

- The database is designed so that it can be consolidated from multiple sources, in which each vulnerability entry includes the origin of information, environment of which it can cause an impact, its consequence, as well as additional useful information.
- The pre-requisites and the consequence of a vulnerability are described using an abstraction which we call a vulnerability expression. The vulnerability expression allows a precise formulation of the nature of the vulnerability and is machine processable. The vulnerability expressions are not specific to a particular system but rather tailorable to the specific system using another mapping, e.g., a configuration file may be mapped separately to its pathname for the particular system being tested.
- The structure of the database should allow easy retrieval both by user and automated-tools via SQL.

We also want to have an automatic scanner which can use the database to do the following:

- Check whether a given vulnerability exists on a local system;

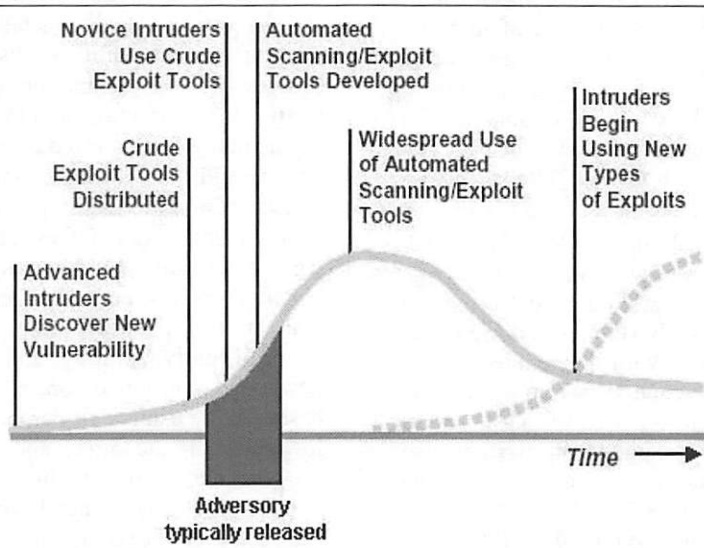


Figure 1: Vulnerability exploit cycle (CERT Coordination Center).

- Scan local system for all possible vulnerabilities;
- Notify the existence of potential vulnerability on the local system should certain environmental factors such as system services which are currently off get activated;
- Analyze relationship between different vulnerabilities, e.g., whether one vulnerability can be exploited to lead to another.

Related Work

There have been a number of popular tools that scan for any presence of vulnerability or configuration weaknesses in a system. Some notable examples are: COPS [4], SATAN [5] and Nessus [6]. These tools are code-based scanning applications where the logic of vulnerability checking is embedded tightly in the scanner's code. This means that including a new check for a vulnerability entry requires one to update the scanner's code, its sub-component(s), or its configuration file. In contrast, our system uses a generic scanner which makes use of vulnerability descriptions stored separately in a vulnerability database. While a code-based solution is generally more powerful, it requires that code/plug-ins be written. There are trust and verification issues which we discuss later in this section.

There is some existing work which reorganizes and integrates information in existing vulnerability databases into one that is more of a "real database." NIST has developed ICAT [7], a searchable index of vulnerability entries leading the users to various vulnerability resources and patch information. Similarly, Purdue University maintains a web-based search system called "Public Cooperative Vulnerability Database" [8]. These databases are, however, designed mainly for vulnerability search based on categorized attribute values, and not for automated applications.

Krsul [9] proposes a comprehensive taxonomy of vulnerabilities for possible further processing or automated manipulation. A database is also proposed. It is hard to compare the database since no specific applications were co-designed with it.

Windows Update [10] is a Microsoft online tool for automatically updating Windows operating systems and Microsoft applications with recent patches. It illustrates some important issues with automatic tools. Windows Update (and its more automatic cousin, Windows Software Update Services [11]) are closed systems. We propose an open system which can cater for heterogeneous environments. Windows Update has a "black-box update model" which allows easy and seemingly automatic patch update, yet the non-transparency of the system leads to the following issues:

1. **Privacy and Trust issues**: As there is no open specification or possible inspection on the scanner, no complete trust can be put on the scanner. This is the case with any code based system. It is difficult to determine if the scanner is performing the correct actions while preserving local system security policy. Since the update system is hosted on the vendor's server, there is no guarantee that local information will not leak to an external source thus breaching local system privacy. In contrast, our system is open to inspection. The database can also be deployed locally or organization-wide as discussed in the deployment section.
2. **Non-standard vulnerability checking**: Windows Update behaves more as a vendor patch update mechanism rather than standardized vulnerability entry checking. Thus, little coverage is given back to users in terms of standard vulnerability report information. This might be too limiting for system administrator who, for example, wants to ensure that his systems are up-to-date against recent vulnerability reports regardless whether patches for the vulnerability are available or not.
3. **No control over the scanner**: Users need to trust that Windows Update works as it is supposed to. The importance of this issue is highlighted by a recent incident of Swen-style Trojan horse which posed as a legitimate update [12]. While this example is a social-engineering style attack, it illustrates the fact that the Windows Update mechanism can itself be a vulnerability. In our system, as the database contains a machine readable description, all steps of the scanner can be verified.

Some related concerns of the Windows Update mechanism is discussed in an article by Berlind [13]. We argue that any automatic update or alert processing mechanism should be based on an open model which can be independently verified. In addition, it should be possible for the user to bypass the automatic system in cases where the security policy may not allow the execution of foreign code or connection to external hosts. Morover, the administrator/user should be able to determine the consequences of a patch or alert on his system.

Movtraq: A New Vulnerability Database

The integrated vulnerability database which we have called *Movtraq (Machine Oriented Vulnerability and Tracking) database* is designed to be compiled from multiple source vulnerability databases and is usable directly by an automatic scanner (see Figure 2).

Design Considerations

The main challenge in designing the new database is to determine what the actual contents of each vulnerability entry should be. For our proof-of-concept, we have focused on what the database should contain rather than on a general database schema. The data fields corresponding to a vulnerability fall into three general categories: general information and references; vulnerability factors and its environmental requirements; and impact of vulnerability.

General Information Fields

The general information portion mostly contains references to several public vulnerability databases such as CERT, Bugtraq, etc. The purpose of these fields is to give the user a reference to the original source of information to obtain additional information. This is mainly for human consumption.

Vulnerability Factors and Environmental Requirements

The second category, vulnerability factors and environment data, provides the main content of machine processable vulnerability information. A vulnerability has to exist within a context, hence it is described in terms of its original source factor and associated environmental factors. By "original source factor," we mean the system component(s) (application or operating system) where the vulnerability originates. "Environmental factors" refers to settings/configuration or services in the local system which make the system subject to the vulnerability.

We distinguish between two kinds of vulnerabilities:

- vulnerability which currently exists on the system; and
- vulnerability which *potentially* exists on the system.

There are a number of different combinations of original source and environment factors:

Case 1: Vulnerability factors: match & Environment factors: match

We will get this result when a particular vulnerability's original source exists on the local system and the settings of local system match all the environment factors. In this case, we will conclude that the vulnerability exists on the system.

Case 2: Vulnerability factors: match & Environment factors: no match

This occurs when we can detect the origin of the vulnerability on the local system, however the settings of the local system does not match the environment factors. So the vulnerability is not applicable but it has the potential to affect the system if the environment changes. For example, consider the case of "Apache Web Server Chunk Handling Vulnerability" [14]. Even if apache is installed, we will not be affected by the vulnerability as long as we do not provide http services.

Although this second case appears to be an exception, it is actually not uncommon as a full installation of the operating system and application programs may have been done. Hence, many installed components in the system may not usually be in use.

Case 3: Vulnerability factors: no match & Environment factors: match

In this case, the vulnerability would appear to be not applicable. However, there is a subtle issue. Consider the case of OpenSSL (an open source implementation of the SSL protocol) which had several stack overflow vulnerabilities which are exploitable [15]. OpenSSL may not be installed as an individual component, so even if there is a database entry for the OpenSSL vulnerability, this would return a negative result in terms of vulnerability data factors. However, OpenSSL is commonly included in applications such as Apache, Sendmail, Bind, Linux and Unix based systems. Thus, it is necessary to check for the existence of such applications which may indicate that such an OpenSSL vulnerability exists even if OpenSSL is itself not detected. This highlights that one may need several database entries corresponding to a vulnerability given some of these indirect potential factors.

Case 4: Vulnerability factors: no match & Environment factors: no match

The vulnerability does not exist on the local system.

Vulnerability Impact (Consequences)

The third category of data concerns the impact of vulnerability, which describes the possible consequences of a vulnerability if it is successfully exploited. In our database, this is stored as a vulnerability description expression which is machine processable and describes the vulnerability impact in a precise and concise form. There is no need to use any taxonomy or qualitative impact factor (e.g., critical, high, medium,

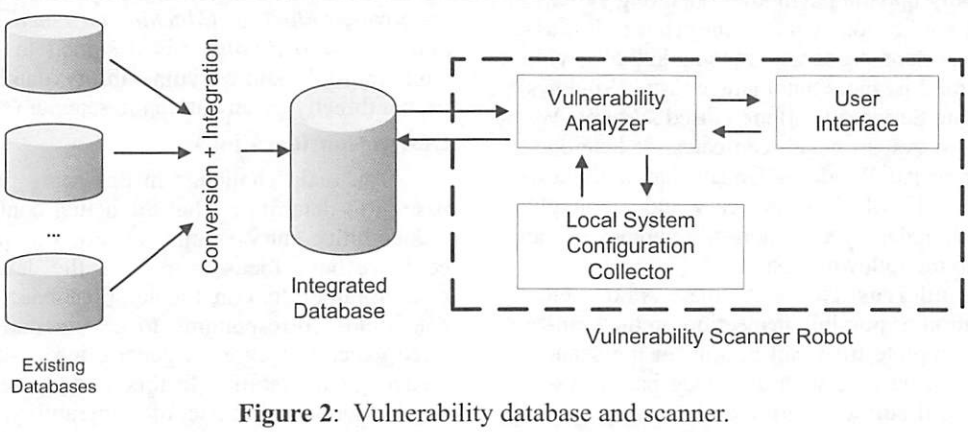


Figure 2: Vulnerability database and scanner.

low) which is not precise and may not make sense in the context of a particular system. It also enables checking of the relationship between different vulnerabilities and whether they can affect one another.

Database Structure

As we have argued, the exact structure of the database is not so important. Rather, it is the content and having it in a more precise machine processable format. In our proof-of-concept design, the database has seven main entities namely:

- Vulnerability Entity – names the vulnerability and links it to the specification of the vulnerability and environment.
- Vulnerability Specifications Entity – collects the vulnerability factors and the impact.
- Environment Specifications Entity – collects the environmental factors.
- Operating System Entity – vulnerability requirements originating from the operating system.
- Application Entity – vulnerability requirements specific to an application.
- Services Entity – vulnerability requirements specific to a service.
- Exploit Entity – details of exploits and impact.

An entity relationship diagram which gives an overview of the relationship between these data items is given in Figure 3.

We will briefly mention some of the key fields from an integration and machine processable perspective. We have mainly omitted fields in the general information category which are present in the database for human consumption.

- **Vulnerability Entity** – a textual description for the vulnerability, identifiers such as CERT ID, BugTraq ID, CVE ID and also other keys corresponding to other tables.
- **Vulnerability Specifications Entity** – vulnerability consequences*, hardware requirements, name of vulnerable application/service*. A service could be a daemon.
- **Environment Specifications Entity** – existence of required user/application or service object* which may be exploitable, existence of a file object*, remote exploitation flag, application/services environment, hardware requirements.
- **Application/Services Entity** – name of application/service, application/service ID, vulnerable versions, hardware requirements. Services have additional fields like protocols, port numbers, etc.
- **Operating System Entity** – similar to application entity but for the operating system.
- **Exploit Entity** – actual exploit (could be a URL, filename, etc.), privileges needed*, consequences of using the exploit*.

The fields which have been labeled by (*) make use of the vulnerability description expressions or vulnerability target objects from the next section. Note that some fields which have a similar function occur a few times in a different context, e.g., hardware requirements may be different for the application and environment, there are two different consequences – one from the vulnerability and one from using a specific exploit.

Integrating the Data

One of the difficulties with dealing with security/vulnerability alerts is the need to integrate the information from multiple sources. Our prototype database

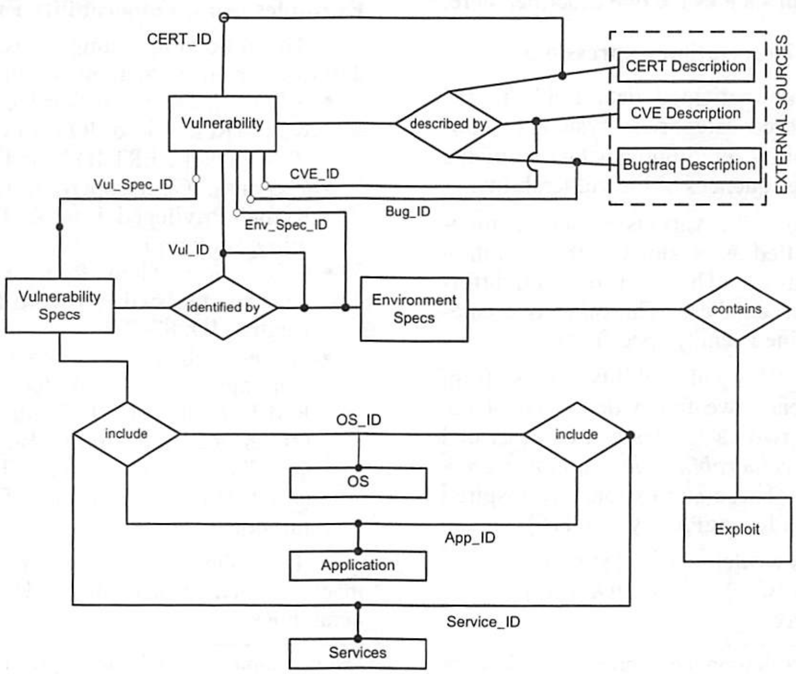


Figure 3: Vulnerability database structure.

is no exception and was built by integrating data from multiple vulnerability sources such as CERT, BugTraq, CVE, vendors and software developer sites. Ideally, one would prefer a single source for the vulnerability information (even if it is only in text form). However, the reality is that due to the distributed handling and speed of dealing with vulnerabilities, one has to accept that integration may be required.

The following example, which is the "OpenSSL SSLv2 Malformed Client Key Remote Buffer Overflow Vulnerability," illustrates the need for integration. It has a CVE ID of CAN-2002-0656 [15].

BugTraq from SecurityFocus provides:
BugTraq ID: 5363
Application environment: Apache v1.0 - 1.3.26
OS environment: Linux, Microsoft Windows
Proof of concept exploit: available
Minimum user rights for exploit: u#R[1]

CERT vulnerability advisory provides:
CERT ID: CA-2002-23
Vulnerable application version:
OpenSSL prior to 0.9.6
Vulnerability impact: @G u#S

Vendor/software information:
From OpenSSL (www.openssl.org) we get the vulnerable application range as: 0.9.1c - 0.9.5a.
From apache documentation we know that usually the user is root.

In general, determining the complete environmental requirements and the consequences of the vulnerability from the textual descriptions can be a tedious and time consuming process. This is one rationale for a better system such as the one described here.

Vulnerability Description Expressions

The main machine oriented data fields in the database belong to three categories: system components of the vulnerability; environment factors of the vulnerability; and consequences of the vulnerability.

The first category for various system components is usually specified as versions of the operating systems and applications. This can be straightforwardly encoded in the database. The other two categories require a machine friendly specification.

After studying 943 vulnerability notes from CERT advisory database, we found that most of the information for these two categories can be described effectively using the *vulnerability description expression* described below. These expressions are inspired by the rule language in KuangPlus system [16].

An expression is written with the syntax:
⟨*VulnerabilityExpression*⟩ = ⟨*TargetObject*⟩ | ⟨*Action*⟩⟨*TargetObject*⟩

An action is written prefixed by '@'. Table 1 illustrates the actions and the types of the corresponding target objects.

Syntax	Semantics
@G ⟨*u*|*g*⟩	Gain ⟨*user object u* | *group object g* ⟩
@R|W ⟨*f*|*m*⟩	Read|Write ⟨*file object f* | *memory object m*⟩
@A ⟨*f*|*m*⟩	Access (read and write) ⟨*file object f* | *memory object m*⟩
@C ⟨*f*⟩	Create ⟨*file object f*⟩
@K ⟨*f*|*m*⟩	Corrupt ⟨*file object f* | *memory object m*⟩
@X ⟨*f*|*c*⟩	Execute ⟨*file object f* | *code object c*⟩
@S|I ⟨*n*|*a*⟩	Crash | Disrupt ⟨*node object n* | *application object a*⟩
@D ⟨*n*|*a*|*s*⟩	Deny ⟨*node object n* | *application object a* | *service object s*⟩
@U ⟨*r*⟩	Use ⟨*resource object r*⟩
@E ⟨*r*⟩	Exhaust ⟨*resource object r*⟩

Table 1: Actions in vulnerability description expressions.

Rather than giving a formal definition of target objects, we have listed examples of target objects in Table 6. In Table 6 the following prefixes are used: '%' is used to denote an actual value; '#' is used to denote a symbolic value; and '&' is used for expressing users/groups associated with an application/service.

As our proof-of-concept implementation is for Unix systems, the examples and objects are also Unix based. Vulnerabilities for other operating systems may require extension to the types of target objects and actions.

Examples using Vulnerability Expressions

The following examples use the expressions to describe various vulnerability consequences.[2]

- @D n#N: Denial of Service for the whole network (Ref: Cisco IOS Interface Blocked by IPv4 Packet CERT ID VU#411332).
- @G u#S: Gain superuser rights. (Ref: Linux Kernel Privileged Process Hijacking Vulnerability, Bugtraq ID: 7112).
- @G u#R : Gain Remote user right (Ref: Apache htpasswd Password Entropy Weakness, Bugtraq ID: 8707).
- @S a#mailman; @D a#mailman: Crash mailman application, and deny its service. (Ref: Red Hat Linux GNU Mailman Remote Denial Of Service Vulnerability, Bugtraq ID: 10147).
- @R f%/etc/passwd : Read file /etc/passwd.
- @X f#*(4777) : Execute a file with setuid permission.

The following are examples of portions of the machine oriented fields in the database for several vulnerabilities:

[1]This is a vulnerability description expression to describe the impact, see the next section.

[2]For simplicity, multiple expressions are separated by semicolon.

Syntax	Semantics
⟨u⟩:	**User Objects**
u#R	Remote user
u#L	Local user
u#S	Super user
u#*	All users
u#P	Physical user
u%100	User with UID 100
u%nobody	User 'nobody' on the system
u#U	User whose privilege is beyond that of the current user
u&App	User running corresponding application process
u&Svc	User running corresponding service (i.e., daemon)
u&Kernel	User who can access or control the OS kernel
⟨g⟩:	**Group Objects**
g#*	All groups
g&⟨App|Svc⟩	Group of the corresponding application process/service
g%50	Group with GID 50
g%sys	Group 'sys' on the system
⟨f⟩:	**File Objects**
f#*	All files
f#passwd	Pathname corresponding to the passwd file
f#shell	Pathname corresponding to shell files, e.g., "/bin/bash"
f#system	Pathname corresponding to system files in the OS

Syntax	Semantics
f#*(4777)	All files with permission 4777
f#F	Files beyond current user access rights
f%/etc/passwd	The file "/etc/passwd"
f&App	File associated to the running application process
⟨m⟩:	**Memory Object**
m#M	Memory area beyond the current user's access right
⟨n⟩:	**Node Objects**
n#S	Scanned node where an application program is installed and/or related service is running
n#L	Nodes in local area network
n#N	Network
n%IP	Node at IP address (may be a range)
⟨a|s⟩:	**Application/Service Objects**
a#AppName	Application Name
s#SvcName	Service Name
⟨r⟩:	**Resource Objects**
r#M	Memory
r#CPU	CPU
r#B	Network Bandwidth
r#D	Disk Space
⟨c⟩:	**Code Object**
c#(⟨u⟩)	Piece of code with the execution privilege of the user object *u*, e.g., privilege escalation

Table 2: Objects in vulnerability description expressions.

- *MySQL Password Handler Buffer Overflow Vulnerability*:
  CVE_ID: CAN-2003-0780
  Bugtraq_ID: 8590
  Vul_Con: @X C#(u%mysql); @G u%mysql; @G u#L
  Vul_OS: null
  Vul_App: Various Mysql versions
  Env_User: u#L
  Env_File: null
  Env_Rem: No
  Exploit: No
  Env_OS: null
  Env_App: Mysql
- *Linux Kernel IOPERM System Call IO Port Access Vulnerability*:
  CVE_ID: CAN-2003-0246
  Bugtraq_ID: 7600
  Vul_Con: @A f#F
  Vul_OS: Various Linux distributions
  Vul_App: null
  Env_User: u#L
  Env_File: null
  Env_Rem: No
  Exploit: No
  Env_OS: Linux kernel 2.4.0 - 2.4.21, 2.5.0 - 2.5.69
  Env_App: null
- *Linux 2.4 Kernel execve Race Condition Vulnerability*:
  CVE_ID: CAN-2003-0462
  Bugtraq_ID: 8042
  Vul_Con: @A f#F; @X c#(u#S); @G u#S
  Vul_OS: Various Linux distributions
  Vul_App: null
  Env_User: u#L
  Env_File: f#*(4111)
  Env_Rem: No
  Exploit: Yes
  Env_OS: Linux Kernel 2.4.0 - 2.4.21
  Env_App: null
- *Multiple Vulnerabilities In OpenSSL*:
  CVE_ID: CAN-2002-0656
  Bugtraq_ID: 5363

Vul_Con: @X c#(u&App); @G u#L
Vul_OS: null
Vul_App: Various Apache versions and OpenSSL-based applications
Env_User: u#R
Env_File: null
Env_Rem: Yes
Exploit: Yes
Env_OS: null
Env_App: Corresponding service provided by the vulnerable application

Translation Issues

From our experiments in translating text-based vulnerabilities into vulnerability expressions, we encountered the following issues:

- The vulnerability description in the database sources is sometimes rather vague. Some examples are: "could expose sensitive information to local attackers" (Bugtraq ID 8233), "gain access to sensitive information" (Bugtraq ID 9558), or "leads to unauthorized access to attacker-specified resources" (Bugtraq ID 9778). We require a more specific consequence which either means describing it in a catch-all fashion or much more work is required to understand the vulnerability.
- Our vulnerability expression language is designed to capture general expressions at the OS level. It does not express various application specific descriptions, such as: "to access variables outside the Safe compartment" (Perl, Bugtraq ID 6111), or "could compromise the private keys of ElGamal signing key implementation" (GnuPG, Bugtraq ID 9115). To deal with such consequences, these are approximated by translation into the closest vulnerability expressions capturable by our language. In the two examples above, we can rewrite them into: access of memory and files beyond the current user's right, respectively.
- Some vulnerability entries, particularly those of CAN(didate) type, are listed as "unknown consequence" (e.g., Bugtraq ID 10428). Hence, we either have to ignore such entries for the moment, or use a special form to indicate unknown consequences.

Movtraq Scanning Robot

To demonstrate the use of the Movtraq database, we have implemented a prototype automatic vulnerability scanner (called the *Movtraq scanning robot).* The robot runs on two different versions of Unix: Redhat Linux and FreeBSD. This is to demonstrate a degree of platform independence.

The overall structure of the robot together with the database is depicted in Figure 2. The integrated Movtraq database is stored in MySql. The scanner consists of a local system configuration collector which collects information about applications, operating system (which processes are running, which ports are open, hardware details, etc.) and services on the system. Software versions are obtained by using the rpm utility on Redhat and pkg_info utility on FreeBSD. The scanner is written in Perl and queries the MySql Movtraq database using SQL.

The robot has three basic scanning options:

- Vulnerability checking: checks if the system is vulnerable to the vulnerabilities specified in the database (a Case 1 vulnerability).

```
1. Apache Mod_Auth_Any Remote Command
   Execution Vulnerability
Application version check: positive
Service port check: negative
Conclusion: source application is detected,
default port required is not open,
potential vulnerability exists but does not
affect current system configuration

2. Sun One/iPlanet Web Server Vulnerability
   to DOS
Application version check: n/a
Conclusion: source application not detected,
safe from vulnerability

3. Linux Kernel IOPERM System Call
   IO Port Access Vulnerability
OS version check: positive
OS environment check: positive
Conclusion: vulnerability detected!

4. MySQL Password Handler Buffer Overflow
   Vulnerability
Application version check: positive
OS environment check: skipped
Conclusion: vulnerability detected!
```

Listing 1: Sample scanner log.

- Potential vulnerability checking: checks for software vulnerability which exists but the system is not currently vulnerable due to environmental reasons (a Case 2 vulnerability). This can be useful since it may be the case that the system can become vulnerable later, e.g., if a service which was off is turned on.
- Vulnerability with exploit checking: enhances vulnerability checking to see if the listed exploits are directly applicable – this adds the constraints of the exploits into the checking process.

An abbreviated sample log from running the scanner illustrates how application, version and environmental checking is performed; see Listing 1. Only some of the pertinent checks from the log are shown to illustrate the following points:

- **Example 1**: apache vulnerability exists but environment check fails since the required port is not open.
- **Example 2**: no vulnerability since application is not installed.
- **Example 3**: an OS vulnerability so only OS checking is used.
- **Example 4**: vulnerability inherent to MySQL version, OS environment checking is skipped as it is not required.

Vulnerability Chaining Analysis

An interesting use of the scanner is that it can be used to test if existing vulnerabilities can be combined together (chaining) to create more vulnerabilities. This mimics what a hacker might do to take advantage of indirect weaknesses on the system.

Consider the following example which is typical of a privilege escalation attack. Suppose the system has the following two vulnerabilities:

Name: Buffer Management Vulnerability in OpenSSH
Vul_ID: 57
CVE_ID: CAN-2003-0693 Bugtraq_ID: 8628
Vul_Con: @G u#L
Vul_OS: null Vul_App: Openssh apps
Env_Usr: u#R Env_File: null
Env_Rem: Yes Exploit: No
Env_OS: null
Env_App: Service provided by the vulnerable app

Name: Linux 2.4 Kernel execve Race Condition Vulnerability
Vul_ID: 48
CVE_ID: CAN-2003-0462 Bugtraq_ID: 8042
Vul_Con: @G u#S
Vul_OS: Linux Vul_App: null
Env_Usr: u#L Env_File: f#*(4111)
Env_Rem: No Exploit: Yes
Env_OS: Linux kernel 2.4.0 - 2.4.21
Env_App: null

In this example, the scanner discovers that both vulnerability 48 and 57 are present. From Vul_ID: 57 a remote user (u#R) can gain local rights (@G u#L), and this chains onto Vul_ID: 48 which has a local environment requirement (local user: u#L and setuid executable file:f#*(4111)). Thus it discovers that a remote user may be able to exploit the two vulnerabilities to gain local root access.

Chaining analysis illustrates the benefit of a machine oriented approach and the use of vulnerability expressions to analyse relationships between vulnerabilities.

Operating System and Local Configuration Mapping

Because environmental and application vulnerability data are expressed as vulnerability expressions, these abstractions may need to be further refined. In the context of a particular local system configuration, operating system distribution, etc., additional localization may be needed to map the abstractions to concrete objects. One may choose to have additional databases to do this mapping from vulnerability target objects to the actual objects on the system. Our robot prototype does not do this since it has been tested only on RedHat and FreeBSD.

Deployment Strategies for vtraq

The prototype Movtraq system is sufficiently useful to be deployed in a number of ways. Some of the potential scenarios depicted in Figure 4 are:

- **Scenario 1**: Local vulnerability database, local client. Here, each local machine hosts its own database. The Movtraq database is meant to have been downloaded (securely) from another server. This has the advantage that the database is local and thus all operations can be done locally. The disadvantage is that an up-to-date database has to be maintained from every host.
- **Scenario 2**: Organization-wide database, local client. This simply extends scenario 1 to an organizational context where there is an organization-wide database server. Where multiple machines have exactly the same configuration, one may choose to only check on a subset of the machines.
- **Scenario 3**: Internet-based database, local client. Lastly, like in automatic update systems, a database server somewhere on the internet serves as the database repository.

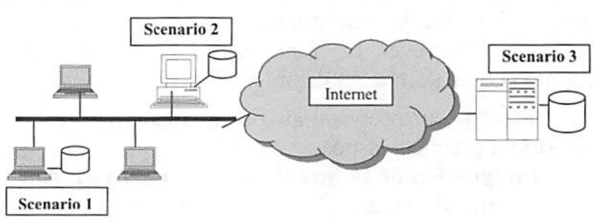


Figure 4: Deployment Options for movtraq.

These strategies are suitable for our Movtraq proof-of-concept system but one could have more

general systems. For example, one could have a scanner which is partially local and partially remote. This may be useful in an organizational context where any system configuration changes are registered with a separate non-local configuration database. Any security alerts are then checked externally against this configuration database.

Discussion

We believe that there is a real need for vulnerability databases which integrate the necessary pieces of information for evaluating the impact of any new vulnerability and allows the appropriate action to be taken automatically. Furthermore, in order to be timely, we argue that the vulnerability evaluation process should not be dependent on having humans process alerts. This does not mean that we advocate not having humans at all in the loop but rather that the loop should not be dependent on the speed of a human response. Thus, it is important that there be a not only human readable vulnerability database but also one which is geared for automatic processing by machines. As far as we are aware, the existing systems for disseminating alerts are still primarily human oriented as are the key source databases.

We have demonstrated a proof-of-concept database which allows effective integration of data from multiple sources and can be used directly by an automatic vulnerability scanner. In the workshop report on security vulnerability databases [17], it was remarked that some of the difficult issues are to do with terminology and the schema of the database. Our database design uses both abstraction and separation of exploits from vulnerabilities – both of which are highlighted in the report. In particular, the use of abstraction, which for us is how the database caters for automated analysis and machine processing, simplifies the issue of terminology and taxonomy. This is a plus point since these are often controversial from a textual description viewpoint.

The database described here is meant to be a proof-of-concept system and is not necessarily comprehensive. However, the prototype scanner demonstrates that we capture the essential elements of a machine-oriented database. As this prototype was designed for Unix systems, for other operating systems, such as Microsoft Windows, both the database and vulnerability expressions may need to be enhanced. However, the fundamental concepts in the design should still be applicable.

Finally, our proposal also addresses a number of important practical issues:

- **Integration of Vulnerability Information**: An integrated database is ideal but may not be practical given that many separate parties are involved in putting together the requisite information. However, it is fairly simple as an additional step to put out the information in the kind of machine oriented form we have advocated and also to concentrate on the relevant data from a machine perspective. In our prototype, we have only built a small integrated database since it is rather time consuming to do so manually from scratch using the existing data sources. However, once vulnerability information is disseminated in the right format, integration becomes significantly easier.
- **Verifiable Vulnerability Processing**: It is certainly the case that any automatic update or scanning system would be welcome by system administrators. However, unless one can deal with the privacy and trust issues, there are significant downsides to the use of such systems. Again, an integrated machine oriented database such as Movtraq allows decoupling of the information from the processing and as it is simply a database, it can be subject to verification.

Further work would involve convenient GUIs, fully featured implementation, Windows compatibility, and a more sophisticated vulnerability model.

Acknowledgments

We acknowledge the support of the "Defence Science and Technology Agency" and "Temasek Laboratories".

Author Information

Sufatrio holds a B.Sc. from University of Indonesia and a MSc from National University of Singapore. He is currently a Ph.D. student in the School of Computing and an associate scientist in Temasek Laboratories, National University of Singapore. His interests include intrusion detection systems and infrastructure for secure program execution. He can be reached electronically at tslsufat@nus.edu.sg .

Roland H. C. Yap obtained his Ph.D. from the Monash University. He is currently an associate professor in the School of Computing, National University of Singapore. His interests include systems security, operating systems, programming languages and distributed systems. He can be reached electronically at ryap@comp.nus.edu.sg .

Liming Zhong graduated from National University of Singapore in 2004. Currently he is working as an IT security specialist in Quantiq International Singapore. His interests cover intrusion detection systems, network and system forensic analysis. Reach him electronically at rick@Quantiqint.com .

Bibliography

[1] CERT Coordination Center, *CERT/CC Statistics 1988-2003*, http://www.cert.org/stats/cert_stats.html , 2003.

[2] CERT Coordination Center, *CERT/CC Overview Incident and Vulnerability Trends*, http://www.

cert.org/present/cert-overview-trends/module-2. pdf, 2003.

[3] Lipson, H. F., *Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues*, CERT Coordination Center, available at http://www.cert.org/archive/pdf/02sr009.pdf, 2002.

[4] Farmer, D. and E. H. Spafford, "The COPS Security Checker System," *Summer USENIX Conference*, 1990.

[5] http://www.fish.com/satan.

[6] http://www.nessus.org.

[7] http://icat.nist.gov/icat.cfm.

[8] https://cirdb.cerias.purdue.edu/coopvdb/public.

[9] Krsul, I., *Software Vulnerability Analysis*, Ph.D. Thesis, Purdue University, COAST technical report 98-09, 1998.

[10] http://windowsupdate.microsoft.com.

[11] http://www.microsoft.com/windowsserversystem/sus/default.mspx.

[12] Keizer, G. "Trojan Horse Poses as Windows XP Update," *TechWeb News*, http://www.informationweek.com/story/show-Article.jhtml?articleID=17300290, 2004.

[13] Berlind, D., "Why Windows Update Desperately Needs an Update," *ZDNet Technical Update*, http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2914519,00.html, 2003.

[14] http://www.cert.org/advisories/CA-2002-17.html.

[15] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0656i.

[16] Howard, J., *Kuangplus: A General Computer Vulnerability Checker*, M.IS. Thesis, Australian Defence Force Academy, 1999.

[17] Meunier P. C. and E. H. Spafford, *Final Report of the Second Workshop on Research with Security Vulnerability Databases*, CERIAS TR 99/06, 1999.

DigSig: Run-time Authentication of Binaries at Kernel Level

Axelle Apvrille – Trusted Logic
David Gordon – Ericsson
Serge Hallyn – IBM LTC
Makan Pourzandi – Ericsson
Vincent Roy – Ericsson

ABSTRACT

This paper presents a Linux kernel module, DigSig, which helps system administrators control Executable and Linkable Format (ELF) binary execution and library loading based on the presence of a valid digital signature. By preventing attackers from replacing libraries and sensitive, privileged system daemons with malicious code, DigSig increases the difficulty of hiding illicit activities such as access to compromised systems.

DigSig provides system administrators with an efficient tool which mitigates the risk of running malicious code at run time. This tool adds extra functionality previously unavailable for the Linux operating system: kernel level RSA signature verification with caching and revocation of signatures.

Introduction

In the past years, the economical impact of malware-like viruses and worms has regularly increased. Even though the target platform of many malware is Windows, with the increasing popularity of Linux as a desktop platform and its wide use as public server, the risk of seeing viruses or Trojans developed for this platform is rapidly growing.

These malware can be installed on the system through different sources. On desktop systems, a major source of malware lies in careless users who introduce viruses, worms, Trojans, or other nuisances through email attachments or internet downloads of Trojaned software.

On server side, very often, vulnerabilities like buffer overflows in public services are exploited by the attacker to install rootkits to replace system binaries and libraries with Trojaned versions. These rootkits are then used to assure a continued access to a compromised machine and mask attacker's illicit activity. For instance, the *Remote Shell Trojan* (RST) infects all ELF binaries of the /bin directory, offering a backdoor process with a command shell at the privilege level of the invoking user.

Even though there are actually different origins to the spread of these malware, the final result is often the same, an unauthorized binary running on the system.

To mitigate this risk, system administrators commonly deploy restrictive solutions such as firewalls, virus scanners, or intrusion detection tools. Although those tools do have positive impact on system security, they have proven to be insufficient; a firewall for instance is usually incapable of detecting covert channels. Papers such as [1] have already raised the alarm, and [2] even compares firewalls to the French Maginot line.[1] Virus scanners and intrusion detection systems also show several limits, such as their incapacity to detect totally new viruses or intrusions. Indeed, their detection engine most usually relies on an extensive signature database, sometimes enhanced with an heuristic algorithm to detect known viruses/intrusions and their close cousins. This results in a time gap between the first spread of new viruses and their characterization through signatures. This gap can be used by attackers to penetrate the internal network of the company. In theory, only few systems based on users' normal behavior or misuse detection model are capable of detecting brand new viruses; however, they are more at research stage yet [5].

Supporting the concept of defense in depth, this paper consequently proposes a new layer of defense to existing mechanisms, named *DigSig*. Used in addition to firewalls, virus scanners, or IDS, this paper highlights how DigSig can significantly increase security. DigSig does not prevent malicious applications to be installed on the system, but prevents their execution, which is when they actually become dangerous.

The paper is organized as follows. First, we describe how DigSig enforces digital signature verification at ELF file loading time. Second, we explain how system administrators would typically deploy DigSig on one or several Linux hosts. Then, we focus on the security aspects of this kernel module and in

[1]For readers not familiar with the history of the second world war, the German army went around the main French defense line, called Maginot and defeated the French army in 40 days!

what way it counters attacks. We analyze the performance impacts of DigSig. Finally, we mention some related work, including some complementary systems DigSig might be added to.

DigSig Kernel Module

DigSig is implemented as a Linux kernel module, which checks that loaded binaries and libraries contain a valid digital signature. In the case of an invalid signature for the binary or for any of its shared libraries, the execution is aborted.

When an ELF file is to be loaded into an executable memory region, DigSig searches the file for a signature section. If no such section is available, loading is refused. Otherwise, DigSig hashes the contents of all text and data segments, and compares the result to the contents of the signature section decrypted with a system's public key. If values do not match, this means the file was not signed with the corresponding private key, or that it was modified after signing. In such a situation, loading is refused.

An attacker who now attempts to replace *ps, ls, sshd, libc*, or any other common rootkit target with Trojaned versions will find these files cannot be executed.

DigSig adds a new section into the ELF binary; therefore, it works only with binaries with ELF format. As ELF is the predominate format in Linux systems, compiling a kernel with only ELF support should not cause problems. In this article, whenever mentioning binaries, only the case of ELF formatted binaries is considered. At this time, DigSig also does not cover the case of scripts. This is outside the scope of this paper, but will be addressed in the future.

The following subsections detail the implementation of DigSig.

Signing the Binary

Before verifying the signature of a binary, the binary needs to be signed and the signature stored in order to retrieve it. Executables and libraries are initially signed offline, using the Debian userspace package *BSign* [3]. BSign embeds an RSA signature of all text and data segments into a new ELF segment called the "signature" segment (see Figure 1). Then, once signed, the RSA private key is safely stored offline and removed from the system, while the corresponding public key is loaded in the DigSig kernel module.

Verifying the Signature of the Binary

At execution time, the DigSig kernel module verifies the signature of the binary/library. This requires the following functionality at kernel level:

- **Support for hash functions**: The first step of each digital signature consists of hashing the data to sign. This is provided by the *CryptoAPI*, which is part of Linux 2.6.x main stream kernels.
- **Public key cryptography**: This is in order to verify the signature.
- **Executable file loading mediation**: This allows us to verify a signature before running the binary, and optionally refuse execution.

The RSA algorithm is used for verifying the signature of the binary. As there is currently no native implementation of RSA at kernel level, we had to import our own implementation into the kernel. Yet, in cryptography, re-inventing the wheel often turns out to be extremely dangerous, so it was decided **to use the well-tested, GPL'ed GPG implementation of RSA** and port the necessary parts for use in kernel space. In order to avoid bloating in the kernel, only 10% of the original code of GPG has been imported. This code is currently isolated in a specific directory (digsig/gnupg) of DigSig.

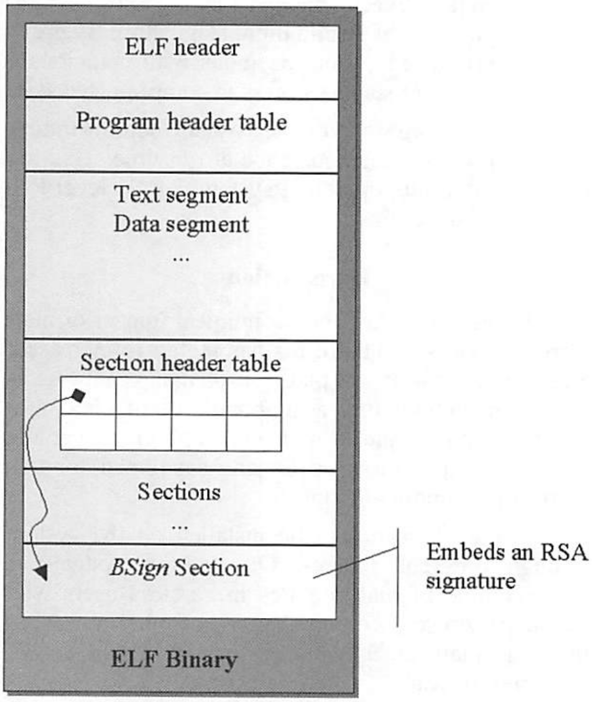


Figure 1: Sign signature section as added in an ELF binary.

As for mediating the loading of executables, DigSig uses the Loadable Security Modules (LSM) architecture [6], which has now become an established part of the kernel. It allows a module to define hooks to annotate kernel objects with security data and mediate access decisions for such objects. One such hook, security_file_mmap, is called whenever mmap is called to map a file to a memory region. This is done by sys_execve to load binaries, as well as by dlopen to load shared libraries. The mmap hook is consequently a convenient location for DigSig to mediate executable mapping of ELF binaries [7].

Caching and Revocation Lists

In order to increase performance of signature verification, DigSig caches a list of binaries whose signatures have already been verified. The first time

an ELF binary or library is loaded, its signature is verified. If the signature is correct, then the successful signature verification is cached. In subsequent loads, DigSig only checks the presence of this signature validation in the cache. This results in a significant improvement in performance, as detailed later. If the file is later opened for writing, the security_inode_permission LSM hook will be triggered, to which DigSig will respond by removing its signature from the cache. The size of the signature verification cache can be specified at module load time, but defaults to 512 signature verifications.

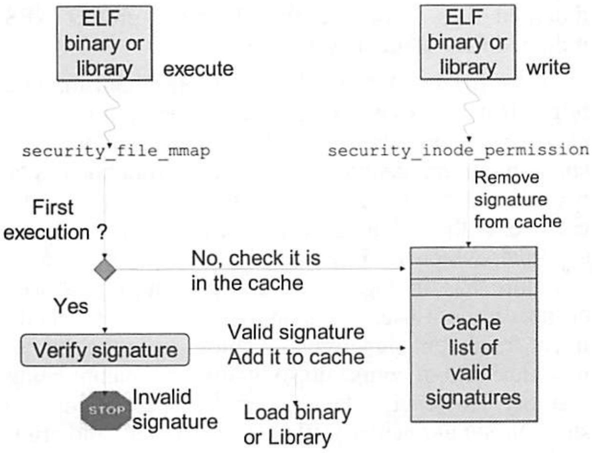


Figure 2: DigSig's caching mechanism.

The introduction of a signature validation cache might seem like a step toward a binary whitelist. In particular, it might seem a simpler solution to eliminate digital signatures altogether, and keep only a whitelist of acceptable binaries and libraries. Files on the whitelist would be executable but not writable, and files not on the whitelist would not be executable. This approach would eliminate the need to verify cryptographic signatures on each file on the first load. However, the DigSig approach has several advantages. First, files with cached signature validations that are not being executed can still be written to. Their signature validation will merely be removed from the cache. Second, the signature validation cache need not be updated to install new libraries or executables. Therefore, software can be installed and upgraded on a live system, and a DigSig system in many ways acts as a normal Linux system. Third, the signature validation cache cannot be edited from user-space, preventing attacks against the cache that an editable whitelist might be subject to. Fourth, a DigSig system can use a relatively small cache, as its sole purpose is performance improvement. A pure whitelist system would need to keep entries for every binary and library.

DigSig also implements a signature **revocation list**, initialized at startup and checked before each signature verification. When programs that were previously signed with the correct private key are later found to contain vulnerabilities, their signature may be revoked by adding them to the revocation list. This is particularly convenient because it eliminates the need to generate a new key pair and resign all binaries and libraries on the system just for a few revocations. The revocation list is communicated to DigSig using the sysfs filesystem, by writing to the /sys/digsig/digsig_revoke file.

Therefore, DigSig allows the binaries to be easily updated over time. This simplifies the evolution of the system over time which is a major requirement for many systems where the binaries can be changed or added over the lifetime of the system.

Deploying DigSig

DigSig requires neither kernel patching nor recompilation of the binaries. Therefore, there are no major changes in the system necessary to deploy DigSig.

DigSig installation requires three initial steps:

1. Generate an asymmetric key pair (for instance using GPG).
2. Sign all trusted binaries and libraries (with BSign).
3. Modify the system startup procedure to load DigSig.

From a deployment point of view, the first step raises the issue of key storage. Obviously the private key should be kept confidential. To this end, Bsign suggests it should be kept on a removable physical support such as a floppy disk or a flash memory key or a read-only CDROM. It is also a good idea to keep a hash of the public key, for instance using sha1sum, to check that an attack hasn't replaced the real public key with another one. This approach is particularly convenient in centralized networks where users connect from remote terminals onto a single application server: DigSig only needs to be deployed on the server to secure execution of all users. However, in networks of individual PCs or laptops, a compromise needs to be made between having one key per host (good security, but perhaps a burden for administrators) and having a single shared key for all (security issue: if one host is compromised, all are). Such issues are not specific to DigSig, but general to PKI.

As for the second step, signing all binaries and libraries can easily be done through the following command:

```
bsign -s -v -I -i / -e /proc \
        -e /dev -e /boot -e
/usr/X11R6/lib/modules
```

It takes about an hour to sign *all* binaries of a Fedora Core 1 installation using Bsign 0.4.5 on a Pentium 4 2.2 GHz with 512 MB of RAM. Note that the revocation list system reduces the need for re-signing the whole system. This fact is particularly important for systems that wish to offer uninterrupted service.

The third step only requires minor modification of the operating system's startup files. Automatic loading of the DigSig kernel module can be achieved by adding DigSig to /etc/modules, and appropriate options in /etc/modutils.

Security Considerations

In this section, a short security analysis of DigSig is conducted to help system administrators understand under what circumstances DigSig does or does not increase security. This assists system administrators in deciding the best approach in deployment.

DigSig operates as a kernel module. It thus requires root privileges for loading and unloading the module, and assumes the secrecy of the DigSig private key, the integrity of its public key, root access to the system, and the Linux kernel itself are not compromised. For the rest of the study, these requirements are taken for granted; however, please note this is not always the case (see the section on related work).

It is important to understand DigSig has not been designed for *vendors* but rather for *system administrators*. System administrators have total control over what should, or shouldn't, execute on the machines they administrate. There is no way a vendor can hope to lock up a given machine to a given software unless with the system administrator's consent. In brief, DigSig targets more prevention against attackers than DRM or software version management. Its two major goals are the following:

1. If a binary has been signed, no one can modify the binary without the modification being detected.
2. Nobody can execute or load an ELF binary or library unless it has been signed.

However, note DigSig cannot protect a system from vulnerabilities within legitimately installed and signed software. Let's see how DigSig achieves security. First, we detail how DigSig prevents modification of ELF binaries. Second, we examine the case of libraries. Finally, we analyze the security of the signature caching and revocation mechanisms.

As described earlier, when a file with a cached signature verification is opened for writing, the signature verification is removed from the cache. However, this does not protect files that are still being executed. Fortunately, the second protection comes from the Linux kernel itself: the kernel forbids executing a file that is opened for writing and reciprocally. This is accomplished by calling deny_write_access(file) kernel hook, and even the superuser is subject to such restrictions.

Unfortunately, the same defense is not extended to shared libraries. Worse, the deny_write_access function is not exported to kernel modules. DigSig must therefore implement its own protection for libraries, which it does very similarly to the kernel. It blocks any attempt to mmap a library with executable permission if that library is already open for writing. If the mmap succeeds, then DigSig increments a usage counter for the inode. So long as the usage counter shows the file to be in use as a library by some process, no one, including the superuser, may open the file for writing.

Under some circumstances, these defenses may still not be sufficient. In particular, deny_write_access (file) and the DigSig shared library writer lock work by marking the VFS inode. They are therefore restricted to a single machine. An NFS mounted file being executed on one client, for instance, could be modified on the server or on any other client. To reduce this threat, DigSig does not cache signature verifications for NFS mounted files. However, this does not protect NFS mounted files while they are in use.

As for the signature caching mechanism, one might fear it introduces a possible attack point. However, since the cache is stored in kernel memory, user space programs cannot directly insert fraudulent signature validations. Signatures may be added to the cache only through a non-exported function dsi_cache_signature, which is only called in one place, when a signature has in fact been validated during dsi_file_mmap. While a user-space application cannot directly inject fraudulent signature validations, a Trojan kernel module could of course do so by directly manipulating memory. However, a Trojan kernel module could also stop DigSig altogether [10], so this is not a valid argument against signature caching.

Finally, it is important to note the signature revocations open the possibility of denial of service. It is vital that an attacker not be able to add *valid* signatures to the revocation list. To ensure this, DigSig restricts access to the communication interface (/sys/digsig/digsig_revoke) to root, so that only root can provide revocation lists to DigSig. As a further precaution, revocation lists can only be appended to before DigSig begins enforcing; that is, before a public key is provided. Care should therefore be taken to ensure that DigSig is enabled before the system is exposed to threats, such as before a network connection is enabled if the network is the primary means of attack. Additionally, the integrity of the collection of revocations must be guarded. For instance, they can be stored on a read-only media (such as a cdrom), or simply signed by GPG.

Performance

Figure 3 presents DigSig's overhead according to the execution time. In the following, all measures are done with a key size of 1024 bits for the RSA algorithm and use of SHA-1. The overhead induced by DigSig grows linearly with the size of executables. However, the gradient is very small: approximately only 0.0016 microsecond per byte.

This is not believed to be critical because unlike Windows operating systems, Unix systems only have few very large executables. As a matter of fact, a typical

Debian Woody workstation only shows 1.8% of executables and libraries above 512 KB.

As DigSig's signature verification is performed once, at the beginning of load time, it is important to note that its induced overhead naturally decreases for long-lived applications. Yet, on Unix systems, administrators and users keep on executing small commands such as *ls*, *cp* and *cd*. In such cases, the cost of signature verification is amortized by DigSig's signature cache (see the second section).

Kernel without DigSig	
real	0m0.004s
user	0m0.000s
sys	0m0.001s
DigSig without caching	
real	0m0.041s
user	0m0.000s
sys	0m0.038s
DigSig with caching	
real	0m0.004s
user	0m0.000s
sys	0m0.002s

Figure 4: Time required for "/bin/ls -Al".

The efficiency of the caching system is demonstrated by Figure 4. This figure displays the average execution time, in seconds, when running a typical texttls -Al command 100 times, using the Unix time command. The benchmark was run on a Linux 2.6.6 kernel with a Pentium 4 2.2 Ghz, 512 MB of RAM. As signature validation occurs in execve, DigSig's overhead is expected to show up during system time (sys). The benchmark results clearly highlight the improvement: there is now hardly any impact when DigSig is used.

Finally, to provide a better insight into the actual impact of DigSig on real workloads, three kernel compiles were timed on a non-DigSig system, and three on a digsig system. The tests were performed using a 2.6.7 kernel on a Pentium 4 2.4 GHz with 512 MB of RAM. The kernel being compiled was a 2.6.4 kernel, and the same .config was used for each compile. Each compile was preceded by a "make clean". Results are shown at Figure 5. The first execution time, both with and without DigSig, appears to reflect extra time needed to load the kernel source data files from disk.

Kernel without DigSig	
real	sys
19m21.890s	1m27.992s
19m 9.276s	1m26.584s
19m 9.464s	1m26.191s
19m 7.717s	1m25.799s
Kernel with DigSig	
real	sys
19m19.957s	1m28.541s
19m 7.485s	1m26.832s
19m 7.883s	1m26.549s
19m 6.494s	1m26.618s

Figure 5: Time required for 2.6.4 kernel "make".

Related Work

This section presents a few related tools that more or less have the same goals as DigSig, but also supplementary work that can be used together with DigSig.

As previously stated, on a security point of view, DigSig assumes the root account has not been compromised. In circumstances where this is unacceptable, there are ways to circumvent this requirement.

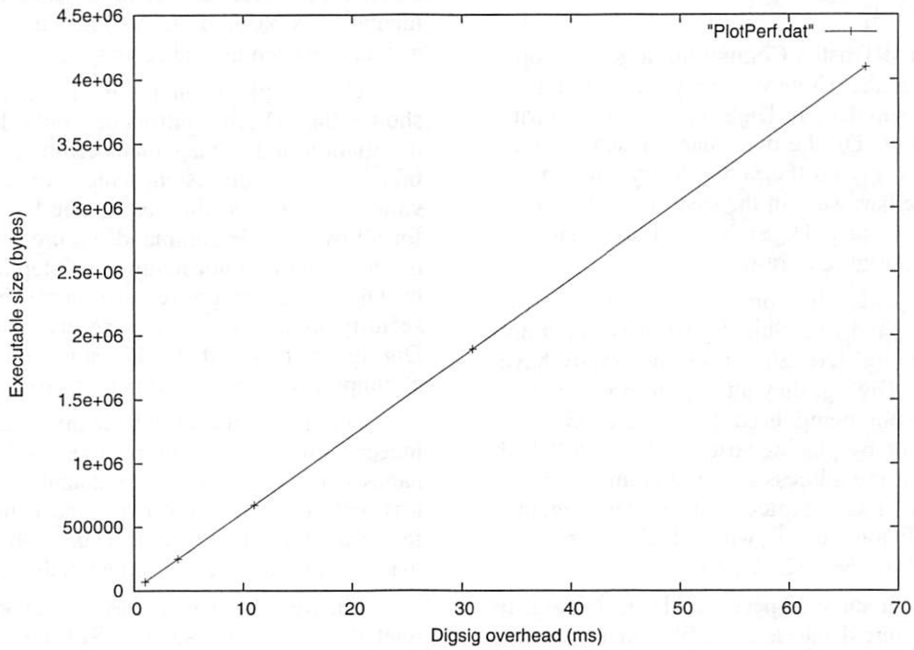


Figure 3: DigSig overhead (ms) for the first load (without caching) per executable size (bytes).

One solution, well known in the Linux domain, relies on SELinux [9]. This security enhanced Linux proposes an implementation of Mandatory Access Control policies, where access control on objects is set according to their sensitivity and not necessarily by their owner. An immediate consequence to this model is that root becomes much less powerful. On "normal" Unix operating systems, root is a super user with super powers. With SELinux, root does not necessarily have access to all objects; this limits risks in case of root compromise. Permissions are specified at a very fine grained level and include access to files, shared memory, POSIX capabilities, and sockets, among others. SELinux could be used in addition to DigSig, to provide the needed secrecy and integrity guarantees of DigSig keys and revocation lists, even in the case of a root compromise.

Another alternative relies on Trusted Computing solutions, such as the specifications provided by the Trusted Computing Group (TCG) [11]. In TCG, normal PC architecture is enhanced with a small security hardware called the *Trusted Platform Module*. In particular, the TPM offers protected storage of data, irremediably binding data to *Platform Configuration Registers* (PCRs). The secret stored on the TPM may only be retrieved by the TPM owner if the configuration of the PC (held in PCRs) hasn't changed. This offers two levels of protection in case of root compromise. First, the TPM owner and root are not necessarily the same person. Second, if root account has been compromised, most of the time this is due to a malicious application (a rootkit) has been installed. Fortunately, installing a rootkit impacts the PC's configuration, so the TPM forbids retrieval of secrets it stores. Sample implementations of trusted computing on Linux can be found in [12, 13, 14].

SELinux and Trusted Computing largely encompass DigSig's goals. However, they offer practical opportunities to supplement DigSig in stricter security sensitive situations. On the other hand, disadvantages of such solutions rely on their complexity, and on the need for specific hardware in the case of trusted computing. On the contrary, DigSig's small size makes it easy to install, configure or re-use.

DigSig may also be compared to similar tools such as PaX [15] and ExecShield [16]. It is important to note that PaX and ExecShield do not exactly have the same goal as DigSig; they attempt to prevent software exploits from being used to execute arbitrary code. This is done by placing strict limits on mmaped regions, and by using address space randomization. In brief, PaX and Exec protect the system against *exploits* of malicious code, while DigSig prevents malicious code from *being executed*.

Tripwire is in some respects similar to DigSig. It maintains a signature database of all files on a machine, and notifies the administrator when some of them are modified (possibly replaced by Trojaned versions for instance). However there are two major differences from DigSig. First, Tripwire works at user level, not within the kernel. Second, it does not provide on-the-fly verification of file signatures. For instance, there is no way to trigger signature verification when a binary is executed. So, Tripwire could more accurately be compared to an off-line file integrity verification tool.

Closer to home, there is a Linux kernel patch written by Greg Kroah-Hartman, from the IBM Linux Technology Center. It is a proof of concept implementing digital signatures in kernel modules. Although it does not check binaries and has no use for caching, it is complementary to DigSig in that the latter does not check Linux kernel modules. The patch modifies a file called module.c, which is responsible for kernel module handling. Unfortunately, LSM does not provide any hooks here. Overall, being a proof of concept, the patch does not benefit from any form of benchmarking or flexibility.

Availability

DigSig is available from SourceForge at http://disec.sourceforge.net. It is available under the GNU Public License version 2. BSign is available with the Debian project, from http://packages.debian.org/unstable/admin/bsign.html.

Conclusion

In this paper, a new tool, named DigSig, is presented. DigSig answers the needs of system administrators in terms of run-time security of Linux operating systems. It focuses on preventing execution of malicious code (ELF binaries or libraries) by checking an embedded RSA signature for each file. The implementation is based on LSM hooks and optimized with a signature caching and revocation mechanism.

On a deployment point of view, the paper has shown that DigSig introduces only little additional installation and management effort. In particular, the initial setup of the system which consists insigning all valid binaries and libraries can be launched once and for all by a single command. Future sporadic changes on the system do not require this step and are handled by DigSig's signature revocation mechanism. From a security point of view, the paper has also demonstrated DigSig is believed to be safe, under reasonable assumptions for most security environments.

DigSig has also been benchmarked, and the results indeed show a very small overhead at load time (a few nanoseconds per byte of executable's size) and even less with the signature caching mechanism. It is therefore reasonable to conclude DigSig should not impact machine's performance from an end-user point of view.

Finally, the paper has presented some other related work. Some, such as SELinux and TCG, may be used in addition of DigSig to supplement it. Others,

such as ExecShield, Tripwire, present similarities and differences that make them suitable for other situations. To our knowledge, at this time, DigSig is the only GPL'ed run-time executable signature verification integrated to the Linux kernel.

In the future, we mainly hope to extend our work to protect Linux systems against malicious shell scripts.

Acknowledgements

The authors would like to thank the LISA reviewers for helpful comments about the extended abstract. They also thank the LISA typesetter, Rob Kolstad, for helpful information as well as for taking the formatting concerns off our hands.

Authors

Axelle Apvrille (axelle.apvrille@trusted-logic.fr) is a senior computer security engineer, currently working for Trusted Logic, in Sophia Antipolis, France.She received her computer science engineering degree in 1996 at ENSEIRB, Bordeaux, France, and then specialized in computer security working at MSI S.A. and in research laboratories of StorageTek and Ericsson Canada. She holds several patents and papers in magazines and international conferences.Her main interests are cryptography, security protocols and embedded security.

David Gordon has a bachelor's degree from the university of Sherbrooke. His interests include security and next-generation networks.

Serge Hallyn graduated from Hope College with a B.S., and the College of William and Mary with M.S. and Ph.D. in computer science. He currently works with the security team at the IBM LTC in Austin, TX. He can be reached at serue@us.ibm.com.

Makan Pourzandi (makan.pourzandi@ericsson.ca) works for Ericsson Research Canada in the Open Systems Research Department. His research domains are security, cluster computing, and component-based methods for distributed programming. He received his doctoral degree on parallel computing in 1995 from the University of Lyon, France.

Vincent Roy is a student in electrical engineering at the University of Sherbrooke. He worked at Ericsson during summer 2004. He can be contacted at gaspoucho@yahoo.com.

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM.

Bibliography

[1] Loscocco, P., S. Smalley., P. Muckelbauer, R. Taylor, S. Turner, and J. Farrell, "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments," *Proceedings of the 21st National Information Systems Security Conference (NISSC)*, October 6-9, 1998.

[2] Frankston, W., *Firewalls: The New Maginot Line*, http://www.frankston.com/public/Essays/Firewalls.asp, February 1998.

[3] BSign Debian package, http://packages.debian.org/stable/admin/bsign.

[4] Apvrille, A., M. Pourzandi, D. Gordon, and V. Roy, "Stop Malicious Code Execution at Kernel Level," *Linux World Magazine*, Vol. 2, No. 1, January 2004.

[5] Sinclair, C., L. Pierce, and S. P. Matzner, "An Application of Machine Learning to Network Intrusion Detection," *Proc. of 15th Annual Computer Security Applications Conference (ACSAC)*, Phoenix, Arizona, 1999.

[6] *Linux Security-Module (LSM) Framework*, http://lsm.immunix.org.

[7] *Tools Interface Standards and Manuals, ELF: Executable and Linkable Format*, http://www.x86.org/intel.doc/tools.htm.

[8] *The Distributed Security Infrastructure Project (DSI)*, http://disec.sourceforge.net.

[9] NSA, *Security Enhanced Linux*, http://www.nsa.gov/selinux/index.cfm.

[10] Truff, "Infecting Loadable Kernel Modules," *Phrack Magazine*, Vol. 11, No. 61, File 10/15, August 2003.

[11] The Trusted Computing Group, http://www.trustedcomputinggroup.org.

[12] Marchesini, J., S. Smith, O. Wild, and R. MacDonald, *Experimenting with TCPA/TCG Hardware, Or How I Learned to Stop Worrying and Love the Bear*, Technical Report TR2003-476, December 15th 2003.

[13] Sailer, R., Y. Zhang, T. Jaeger, and L. van Doorn, *Design and Implementation of a TCG-based Integrity Measurement Architecture*, IBM Research Report RC23064, January 16th, 2004.

[14] Safford, D., J. Kravitz, and L. van Doorn, "Take Control of TCPA," *Linux Journal*, Issue 112, http://www.linuxjournal.com/article.php?sid=6633, August 2003.

[15] Busser, Peter, "Memory Protecting with PaX and the Stack Smashing Protector," *Linux Magazine*, Issue 40, March 2004.

[16] Exec-Shield, http://people.redhat.com/mingo/exec-shield.

[17] Kroah-Hartman, Greg, "Signed Kernel Modules," *Linux Journal*, Issue 117, http://www.linuxjournal.com/article.php?sid=7130, January 2004.

I^3FS: An In-Kernel Integrity Checker and Intrusion Detection File System

Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok
– Stony Brook University

ABSTRACT

Today, improving the security of computer systems has become an important and difficult problem. Attackers can seriously damage the integrity of systems. Attack detection is complex and time-consuming for system administrators, and it is becoming more so. Current integrity checkers and IDSs operate as user-mode utilities and they primarily perform scheduled checks. Such systems are less effective in detecting attacks that happen between scheduled checks. These user tools can be easily compromised if an attacker breaks into the system with administrator privileges. Moreover, these tools result in significant performance degradation during the checks.

Our system, called I^3FS, is an on-access integrity checking file system that compares the checksums of files in real-time. It uses cryptographic checksums to detect unauthorized modifications to files and performs necessary actions as configured. I^3FS is a stackable file system which can be mounted over any underlying file system (like Ext3 or NFS). I^3FS's design improves over the open-source Tripwire system by enhancing the functionality, performance, scalability, and ease of use for administrators. We built a prototype of I^3FS in Linux. Our performance evaluation shows an overhead of just 4% for normal user workloads.

Introduction

In the last few years, security advisory boards have observed an increase in the number of intrusion attacks on computer systems [2]. Broadly, these intrusions can be categorized as network-based or host-based intrusions. Defense against network-based attacks involves increasing the perimeter security of the system to monitor the network environment, and setting up firewall rules to prevent unauthorized access. Host-based defenses are deployed within each system, to detect attack signatures or unauthorized access to resources. We developed a host-based system which performs integrity checking at the file system level. It detects unauthorized access, malicious file system activity, or system inconsistencies, and then triggers damage control in a timely manner.

System administrators must stay alert to protect their systems against the effects of malicious intrusions. In this process, the administrators must first detect that an intrusion has occurred and that the system is in an inconsistent state. Second, they have to investigate the damage done by attackers, like data deletion, adding insecure Trojan programs, etc. Finally, they have to fix the vulnerabilities to avoid future attacks. These steps are often too difficult and hence machines are mostly re-installed and then reconfigured. Our work does not aim at preventing malicious intrusions, but offers a method of notifying administrators and restricting access once an intrusion has happened, so as to minimize the effects of attacks. Our system uses integrity checking to detect and identify the attacks on a host, and triggers damage control in a timely manner.

In our approach, given that a host system has been compromised by an attack, we aim at limiting the damage caused by the attack. An attacker that has gained administrator privileges could potentially make changes to the system, like modifying system utilities (e.g., /bin files or daemon processes), adding backdoors or Trojans, changing file contents and attributes, accessing unauthorized files, etc. Such file system inconsistencies and intrusions can be detected using Tripwire [10, 9, 22]. Tripwire is one of the most popular examples of user mode software that can detect file system inconsistencies using periodic integrity checks. There are three disadvantages of any such user-mode system: (1) it can be tampered with by an intruder; (2) it has significant performance overheads during the integrity checks; and (3) it does not detect intrusions in real-time. Our work uses the Tripwire model for the detection of changes in the state of the file system, but does not have these three disadvantages. This is because our integrity checking component is in the kernel.

In this paper we describe an in-kernel approach to detect intrusions through integrity checks. We call our system $\mathbf{I^3FS}$ (pronounced as *i-cubed FS*), which is an acronym for **I**n-kernel **I**ntegrity checker and **I**ntrusion detection **F**ile **S**ystem. Our in-kernel system has two major advantages over the current user-land Tripwire. First, on discovering any failure in integrity check, I^3FS immediately blocks access to the affected file and notifies the administrator. In contrast, Tripwire checks are scheduled by the administrator, which could leave a larger time-period open for multiple

attacks and can potentially cause serious damage to users and their data. Second, I³FS is implemented inside the kernel as a loadable module. We believe that the file system provides the most well-suited hooks for security modules because it is one level above the persistent storage and most intrusions would cause file system activity.

In addition to providing these advantages over Tripwire, our system is implemented as a stackable layer such that it can be stacked on top of any file system. For example, we can use stacking over NFS to provide a network-wide secure file system as well. Finally, it is easier to compromise user-level tools (like Tripwire) than instrumenting successful attacks at the kernel level.

We used a stackable file system template generated by FiST [28] to build an integrity checking layer which intercepts calls to the underlying file system. I³FS uses cryptographic checksums to check for integrity. It stores the security policies and the checksums in four different in-kernel Berkeley databases [8]. During setup, the administrator specifies detection policies in a specific format, which are loaded into the I³FS databases. File system specific calls trigger the integrity checker to compare the checksums for files that have an associated policy. Based on the results, the action is logged and access is allowed or denied for that file. Thus, our system design uses on-access, real-time intrusion detection to restrict the damage caused by an intrusion attack.

Design

Checksumming using hash functions is a common way of ensuring data integrity. Recently, the use of cryptographic hash functions has become a standard in Internet applications and protocols. Cryptographic hash functions map strings of different lengths to short fixed size results. These functions are generally designed to be collision resistant, which means that finding two strings that have the same hash result is impractical. In addition to basic collision resistance, functions like MD5 [19] and SHA1 [4] also offer randomness, unpredictability of the output, etc. In I³FS, we use MD5 for computing checksums.

We have designed I³FS as a stackable file system [26]. File system stacking is a technique to layer new functionality on top of existing file systems, as can be seen in Figure 1. With no modification to the lower level file system, a stackable file system operates between the virtual file system (VFS) and another file system. I³FS intercepts file system calls and normally passes them to the lower level file system; however, I³FS also injects its integrity checking operations and based on return values to system calls, it affects the behavior that user applications see.

When designing I³FS, we aimed at offering a good balance between security and performance. We offer configurable options that allow administrators to tailor the features of I³FS to their site needs, trading off functionality for performance.

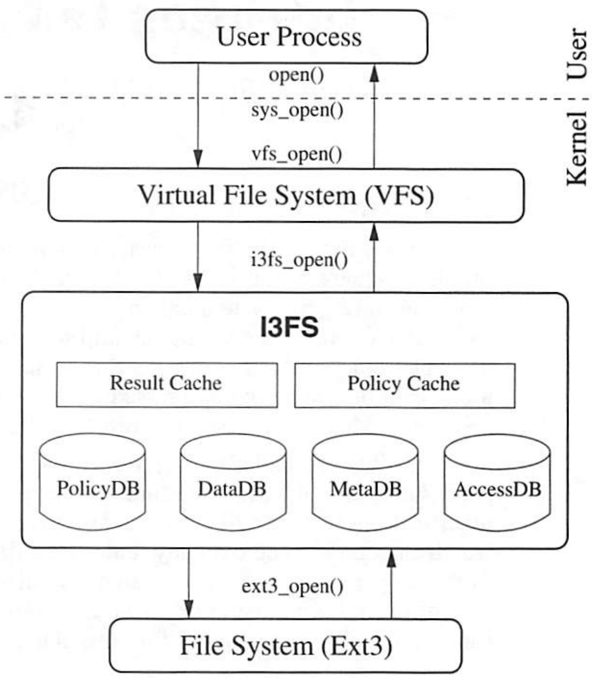


Figure 1: I³FS Architecture.

Threat Model

I³FS is primarily aimed at detecting the following:

- Malicious replacement of vital files such as the ones in the /bin directory. Attackers could replace programs such as ls and ps with Trojans, without the knowledge of the system administrators. These kind of attacks can be tracked and prevented through I³FS by setting up appropriate policies for important files.
- Unauthorized modification of data by an eavesdropper in the network, in the case of remote file systems, where the client file system communicates with the disk over an insecure network.
- Corruption of disk data due to hardware errors. Inexpensive disks such as IDE disks silently corrupt data stored in them due to magnetic interference or transient errors. These errors cannot always be detected by normal file systems. I³FS can notify the administrator about disk corruption if there is a suitable policy associated with the file.

Policies

The two main goals we considered when designing the policies for I³FS were versatility and ease of use. The policy syntax provided by I³FS is similar to the user level Tripwire [10]. The general format of an I³FS policy is as follows:

```
{-o|-e|-x} OBJECT -m FLAGS -p PROPERTIES
-a ACTION [-g GRANULARITY] [-f FREQ]
```

where

- -o *OBJECT* specifies the object (file or directory) for which the rule is valid. If the object is a directory, then the rule applies recursively to all the files and sub-directories. The -e option is used to exclude an object (in most cases sub-directories) from the integrity checks, and the -x option is used to remove a policy.
- -m *FLAGS* represents the set of attributes of the respective object, used to calculate the checksum. The supported attributes are as follows:
  - **p** *Permission and file mode bits*
  - **i** *Inode Number*
  - **n** *Number of Links*
  - **u** *User id of the owner*
  - **g** *Group id of the owner*
  - **s** *File size*
  - **d** *ID of the device on which the inode resides*
  - **b** *Number of blocks allocated*
  - **a** *Access time*
  - **m** *Modification time*
  - **c** *Inode change time*
- -p *PROPERTIES* represents the properties of the policies, used to calculate the checksum. The properties offered are as follows:
  - **D** *Checksum file data*
  - **I** *Inherit the policies for new files*
- -a *ACTION* determines the action taken if the integrity check failed. Our I^{3}FS implementation supports only two actions: BLOCK and NO-BLOCK. The BLOCK action returns a failure for any attempt to access the respective file and alerts the administrator about the inconsistency of this critical resource. The NO_BLOCK action lets the operation go through I^{3}FS to the underlying file system. All integrity check failures are logged in the I^{3}FS system.
- -g *GRANULARITY* specifies whether the checksumming is done on a per page basis or for the entire file at once. The available granularity options are PER_PAGE or WHOLE_FILE. PER_PAGE is useful for mostly-random file access patterns, and WHOLE_FILE is useful for mostly sequential small-file access patterns.
- -f *FREQ* is an integer value that determines the frequency of integrity checks. For example, a value of 50 for frequency would make I^{3}FS perform integrity checking for the file every 50 times it is opened. This option is available only if WHOLE_FILE checksumming is chosen.

We have chosen the set of policy options such that it helps detect most kinds of attacks on the file system. Checksumming different fields of the meta data of files helps detect whether important files have been re-written by malicious programs through the file system. Checksumming file data helps detect unauthorized modification of data possibly made without the knowledge of the file system. An example of this is a malicious process that can write to the raw disk device directly in Unix-like operating systems.

I^{3}FS Databases

I^{3}FS configuration data is stored in four different in-kernel databases. KBDB [8] is an in-kernel implementation of the Berkeley DB [21]. Berkeley DB is a scalable, high performance, transaction-protected data management system that efficiently and persistently stores ⟨key, value⟩ pairs using hash tables, B+ trees, or queues. I^{3}FS stores four databases in the B+ tree format, so that we benefit from locality. The schema for the four databases is given in Figure 2.

Database	Key	Value
`policydb`	inode#	Policy bits, freq#
`datadb`	inode#, page#	Checksum value
`metadb`	inode#	Checksum value
`accessdb`	inode#	Counter value#

Figure 2: I^{3}FS: Database schemas.

Having separate databases for storing the data and meta data checksums is advantageous in certain situations. Generally, we expect that meta data checksumming would be used more commonly than data checksumming for two reasons. First, almost all modifications to a file made through the file system will result in modifications to its meta data. Second, meta data checksumming is less time-consuming than data checksumming as the number of bytes to be checksummed is smaller. Therefore, having the data and meta data checksums in two different databases results in less I/O and more efficient cache utilization as data checksums need not be fetched along with meta data checksums.

The policy database (policydb) contains the policy options associated with the files and optionally the frequency of check values. We use the inode number to refer to the policies instead of the path names so as to avoid unnecessary string comparisons. We have a user level tool that reads the policy file and populates the policy database. Further details about initialization and setup are given later. The policy database has the inode number of the file as the key, and the data is either a 4 byte or an 8 byte value containing the policy bits and optionally the frequency of integrity checks (if the frequency of checks policy option is chosen).

The data checksum database (datadb) contains the checksums of file data for those files that have a policy option for checksumming their data. Since there are two sub-options for checksumming file data, the per page and the whole file checksumming, this database either contains *N* of checksums for a single file, where *N* is the number of pages in the file, or a single checksum for the entire file. The inode number and the page number form the key for this database. If the option is to checksum the whole file, then the single checksum

value will be indexed with page number zero. The data checksum database is populated during the I^{3}FS initialization phase through an ioctl, when the policies are added to the policy database.

The meta data checksum database (metadb) has a simple design. The key is the inode number and the data is the checksum value for the set of fields of the inode that are specified in the policy options. Information about the set of fields that are checksummed is not stored in the meta data checksum database. Instead it is retrieved from the policy database that stores the policy bits. This database is also populated during the initialization phase which we discuss later.

The access counter database (accessdb) contains a counter that represents the number of times a file has been opened after the last time it was checked for integrity. This is useful to set custom numbers for frequency of checks, so that less important files need not be checked for integrity every time they are accessed. For files that have a policy indicating a custom frequency of check number, every read will result in getting the previous counter value, increasing it by one and saving the new value, if the counter has not exceeded the frequency limit.

Caching in I^{3}FS

In I^{3}FS, each file access is preceded by a check whether that file has an associated policy or not. We expect that the number of files that have policies will be much less than the number of files without policies. Hence, it is important that we optimize for the common case of a file without an associated policy. Second, for those files that have policies and are accessed frequently, checksums should not be re-computed on each access.

I^{3}FS caches two kinds of information. First, whether a given file has a policy associated with it or not, and if so, the policy for that file. Second, it caches the result of the previous integrity check. All information is cached in the private data of the in-memory inode objects. The inode private data includes several new fields to cache policies, meta data checksum results, whole file data checksum results, and per page checksum results. While caching the policies, the result of the check for the existence of a policy is also cached. This mechanism serves the purpose of having both a positive and negative cache for the existence of policies, thereby expediting the check for both files that have and those that do not have policies associated with them.

As a per page integrity checking cache, the inode private data contains an integer array with ten elements which acts as a page bitmap. The cache can hold the integrity check results for the first 320 pages when run on an i386 system with page size of 4KB. Thus the results for files which are less than 5 MB can be fully cached. This accounts for almost 90% of the files in a normal system [7].

The data and meta data integrity check result caches are invalidated every time there is a data or inode write for that file. Since all information is cached in the inode private data and not in an external list, the reclamation process for the inode cache will take care of the I^{3}FS configuration cache reclamation also. This method of caching is advantageous because the inodes for the frequently-accessed files will be present in the inode cache and hence the policies and results for those files will also be present in the cache.

Securing I^{3}FS Components

Securing the databases that store the configuration and setup of I^{3}FS is one of the prime requirements for making I^{3}FS a secure file system. There can be valid updates to the checksums needed when a file needs to be genuinely updated and these updates have to follow a secure channel so that there can be a clear differentiation between authorized and unauthorized updates to the files. I^{3}FS uses an authentication mechanism to ensure that updates to the checksums are made by authorized personnel.

I^{3}FS stores the four databases that it uses in an encrypted form. We use the in-built cryptographic API provided by the Berkeley Database Manager for encryption. We use the AES encryption algorithm [14] with a 128-bit key size. Since I^{3}FS requires a key to be provided for reading the encrypted database, we wrote a custom file system mount program that accepts the passphrase from the administrator. Having the database encrypted prevents unauthorized reading of the database file without going through the authentication process.

Authentication

An authentication mechanism is required for I^{3}FS for two reasons. First, mounting and setup of I^{3}FS should be done through a secure channel so that malicious processes that acquire super user privileges could not mount the file system with incorrect configuration options. Second, valid updates to the files that carry policies should be permitted only through a secure channel. This is because critical programs and files need to be updated occasionally and such updates should not require reinitialization of the file system.

Since the I^{3}FS databases are encrypted, reading them requires a passphrase. Therefore we provide a custom mount program that authenticates the person mounting the file system. The first time I^{3}FS is mounted, the administrator is prompted for a passphrase. This passphrase is used to compute the cryptographic hash for a known word, "i3fspassphrase." We store this hash as part of the policy database. During subsequent mounts, the passphrase entered is validated by computing the hash again and comparing it with the stored hash. Upon mismatch of hashes, the mount process is aborted and an error message is returned to the user-level mount program. If the passphrase entered is correct, it is stored in the private data of the super-block structure. Thus the passphrase is kept non-persistent and stored in the kernel memory only.

The checksums for all files that have a policy are computed during the initialization phase of I^{3}FS. To allow valid updates to files whose checksums have already been stored, we provide two modes of operations for I^{3}FS: one that allows updates and another that does not. This is implemented using a flag in the in-memory super block which can be set and unset from user level through an ioctl. This ioctl can be executed only after providing a valid passphrase. The passphrase passed to the ioctl is compared with the one that is stored in the super block private data and access is granted based on the result. A similar authentication method is implemented for the ADD_POLICY and REMOVE_POLICY ioctls as well.

Actions Upon Failure

There are two kinds of actions that can be specified for files for which integrity checking fails. They are the BLOCK and NO-BLOCK options. The BLOCK option disallows access to files that fail integrity check, and a message is recorded in the log. In the case of NO-BLOCK option, access is allowed for files that fail integrity check but an appropriate message is logged. By default I^{3}FS logs messages through syslog. Optionally, a log file name can be given as a mount option to the custom mount program, and all log messages will be written directly to that file.

Implementation

I^{3}FS is implemented as a stackable file system that can be mounted on top of any other file system. Unlike traditional disk-based file systems, I^{3}FS is mounted over a directory, where it stores the files. In this section we discuss the key operations of I^{3}FS and their implementation.

Initialization and Setup

The first time the administrator mounts I^{3}FS, a passphrase needs to be entered for the file system to initialize itself. The first mount operation will store the HMAC hash of a known word, "i3fspassphrase," hashed using the passphrase entered, into the policy database for authenticating further mounts. After the file system is mounted, the administrator has to run a user level setup tool that takes a policy file as input. The format of the policy file is described in the section on design policies. The user level utility calls the corresponding ioctls to set up the policies to the four databases:

- ADD_POLICY: This ioctl takes the passphrase, path name, and policy bits (an integer) as input. It verifies the passphrase, converts the path name to an inode number, and stores the inode number and the policy bits in the policy database. In addition, based on the policy bits, the ioctl computes meta data and data checksums appropriately and inserts them into the meta data and data databases.
- REMOVE_POLICY: This ioctl takes a passphrase and path name as input. It authenticates and then converts the path name to an inode number and removes all entries from all four databases that match the given inode number.
- ALLOW_UPDATES: This ioctl takes the passphrase as argument. It just authenticates and sets the AUTO_UPDATE flag in the in-memory super block which allows updates to files with a policy, along with the update of their checksum values.
- DISALLOW_UPDATES: This ioctl resets the update flag in the super block private data, so that further updates to checksums of files with policies are stopped.

Recursive policies can also be specified for directories, so that the policy is applied to all files inside the directory tree. In this case, the user level program uses nftw(3) to enumerate the set of files for which the policies should be applied. It then calls the ioctl for each of the files.

The usage of the user level program i3fsconfig is as follows:

```
i3fsconfig [-u ALLOW|DISALLOW]
           [-f POLICYFILE]
```

Mount Options

I^{3}FS is mounted using a custom mount program that uses the mount(2) system call. It uses getpass(3) to accept a passphrase from the administrator and passes the passphrase as a mount option. A custom mount program is used instead of the Linux mount program because the passphrase entered should not be visible at the user level after the file system is mounted.

There are three optional mount parameters.

- The auto-update option sets the AUTO_UPDATE flag in the kernel so that checksums will be updated every time a file with a policy is updated.
- The logfile option allows one to specify a separate log file where the I^{3}FS log messages can be written to.
- The dbdir option allows the administrator to set the location of the checksum databases. Normally these databases are stored inside the file system, and I^{3}FS prevents direct access to them and hides them from view. With this option, administrators can place the databases in a different directory than the checksummed file system; this is useful, for example, when I^{3}FS is stacked on top of NFS because the databases could be kept on a safer local directory.

Meta Data Integrity Checking

The flowchart for meta data integrity checking is shown in Figure 3. For checksumming the meta data, since we have a customizable set of inode fields to be checksummed, we need to use both the policy bits and the stored checksums for integrity checking. The meta data integrity checking is done in the file permission check function, i3fs_permission. The i3fs_permission function is called after lookup for every file that is accessed. Hence, the integrity check cannot be bypassed

for files with a policy. The permission check function first checks the policy cache to determine if the file's inode has a policy associated with it. Upon a cache miss, it refers to the policy database and then decides the result. If there is a policy associated with the file, its policy bits are retrieved from the policy database. From the policy bits, the set of inode fields that have to be checksummed is found and the checksum for those fields is computed. This computed checksum is verified against the checksum value stored in the meta data checksum database. If both checksums match, then access is granted; if the checksums do not match, then the necessary action is performed as per the action policy bit.

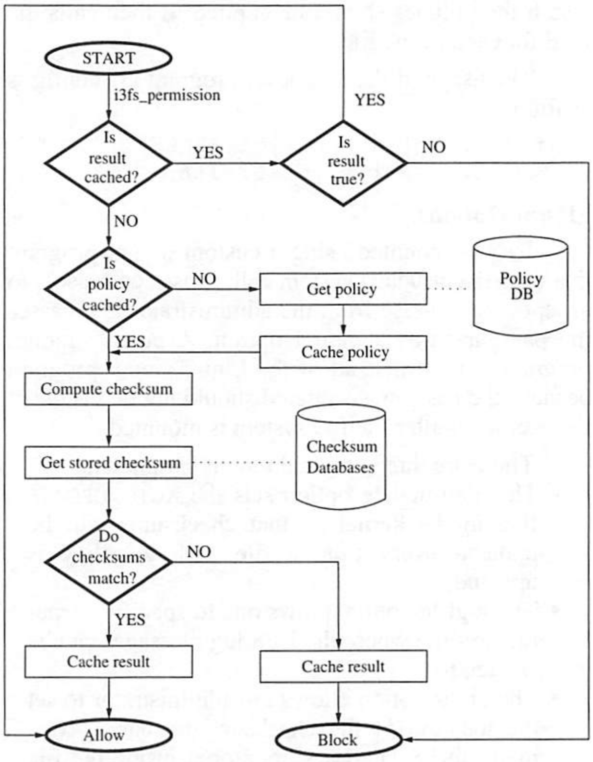


Figure 3: Flowchart for I³FS permission checks.

Once it is determined if the inode has a policy associated with it or not, the information is stored in the in-memory inode as a cache for further accesses. As long as that inode is present in the inode cache, the policy information will also be cached.

Data Integrity Checking

We provide two options for checksumming file data. The first is PER_PAGE checksumming and the second is WHOLE_FILE checksumming. In the case of per page checksumming, integrity checking is done in the page level read function, i3fs_readpage. If the option is to checksum whole files, then the integrity checking is done in the open function, i3fs_open. In the case of WHOLE_FILE checksumming, whenever i3fs_open is called for a file with a policy, the policy bits present in the in-memory inode are checked. If the data checksum mode bit is set to WHOLE_FILE, then the checksum for the whole file is computed and verified against the checksum value stored in the data checksum database. If they match, i3fs_open succeeds; if not, the necessary action is performed as per the policy bits. The WHOLE_FILE integrity check results are cached in the inode private data in a field named whole_file_result.

In PER_PAGE integrity checking, during i3fs_readpage, I³FS checks the policy bits to determine whether page-level checksumming is enabled. If yes, then the checksum is computed for that page and it is compared with the stored value in the data checksum database. The result is cached in the page bitmap present in the inode private data as explained later.

Frequency of Checks

Since whole file checksumming is a costly operation, we provide an option for specifying the frequency of integrity checks in the policy. For performance reasons, one can set up a policy for a file such that it will be checked for integrity every N times it is opened, where N is an integer value. Every time a file with a policy is opened, we check if it has a frequency number associated with it. If yes, the counter entry for the file in the access database is incremented by one. When the value is equal to N, integrity check is performed and the counter is then reset to zero.

Updating Policies

Policies that are enforced when the file system is initialized might not remain valid at all times. We provide a method by which the administrator can update the policies dynamically without reinitializing the system. This can be done using the following two ioctls: ADD_POLICY and REMOVE_POLICY. The administrator can either add policies to new files or remove policies from existing files. If a policy for an existing file has to be modified, it has to be first removed and then re-inserted using the ADD_POLICY ioctl.

Updating Checksums

Often it is required that files with policies be updated from time to time. For example, administrators need to install or upgrade system binaries. Such updates should also re-compute the checksums that are stored in the databases, so that I³FS need not be reinitialized for every file update. However, these kinds of checksum updates should be allowed through a secure channel so as to prevent malicious programs from triggering checksum updates subsequent to an unauthorized modification to file data. In I³FS, we provide a flag called AUTO_UPDATE which can be set and reset by the administrator after authenticating using the passphrase. This can also be set during mount as a mount option. When the AUTO_UPDATE flag is set, all updates to files with policies will update the checksums associated with them. If the flag is not set, file data updates will be allowed without updates to the

checksums so that these are categorized as unauthorized changes. The AUTO_UPDATE flag can only be set from a console for security reasons; processes that are executing in a non-console shell are not allow to update checksums when the AUTO_UPDATE flag is set.

The checksum updates for meta data are done in the put_inode file system method of I^{3}FS. Whole file checksums are updated in the file release method and per page checksums are updated in the writepage and commit_write methods, respectively.

Inheriting Policies

To facilitate automatic policy generation for new files that get created after I^{3}FS is initialized, we provide a method for the policy of the parent directory to be inherited by the files and directories that are created under it. This can be used by setting the INHERIT policy bit for the directory in question. Whenever a file or a directory is created, the policy of the parent directory is copied for it, if its parent directory has the INHERIT bit set. However, for checksums to be updated for the new file, the AUTO_UPDATE flag must be set. If the flag is not set, then the policy of the parent directory will be copied for the new file, but the checksums will not be updated. Thus the next time such a file is accessed there will be a checksum mismatch.

Evaluation

We used the stackable templates generated by FiST [28] as our base, and it started with 5,670 lines of code. To implement I^{3}FS, we added 4,227 lines of kernel code and 300 lines of user level code. In addition to this, I^{3}FS includes 367 lines of checksumming code implemented by Aladdin Enterprises [5]. We wrote two user level tools: a custom mount program for I^{3}FS and another tool for setting up, initializing, and configuring I^{3}FS. I^{3}FS is implemented as a kernel module and requires the in-kernel Berkeley database [8] module to be loaded prior to using I^{3}FS.

To measure the performance of I^{3}FS, we stacked I^{3}FS on top of a plain Ext2 file system and compared its performance with native Ext2. All measurements were conducted on a 1.7 GHz Pentium 4 with 1 GB RAM and a 20 GB Western Digital Caviar IDE disk. For the frequency of checks experiment, we lowered the amount of memory to 64 MB. The operating system we used was Red Hat Linux 9 running a vanilla 2.4.24 kernel. We unmounted and remounted the file systems before each run of benchmarks so as to ensure cold caches. All benchmarks were run at least ten times and we computed 95% confidence intervals for the mean elapsed, system, user, and wait time using the Student-*t* distribution. In each case, the half widths of the intervals were less than 5% of the mean. In the graphs in this section, we show the 95% confidence interval as an error bar for the elapsed time. Wait time is the elapsed time less CPU time and user time and consists mostly of I/O, but process scheduling can also affect it.

We calculated the overheads of I^{3}FS under several different configurations and a variety of system workloads. Based on the types of policies, we classified the tests as follows:

- Without any policies (NP)
- Only meta data checksumming (MD)
- Meta data and whole-file data checksumming (MW)
- Meta data and per-page data checksumming (MP)
- Meta data and whole-file checksumming with inheritable policies (MWI)
- Meta data and per-page checksumming with inheritable policies (MPI)

Each of the above configurations of I^{3}FS are used to identify the isolated overheads of the components of I^{3}FS. The NP configuration does not compute checksums, and is useful in finding the overheads due to the stackable layer and to check whether files have policies associated with them or not. The MD configuration is used to find the overhead of checksumming the meta data alone. The other configurations, MW, MP, and MPI, isolate the overheads associated with each of the checksumming options described in the design section.

We tested I^{3}FS using a CPU-intensive benchmark, an I/O-intensive benchmark, and a custom read benchmark to test the frequency of checks performance.

For a CPU-intensive benchmark, we compiled the Am-utils package [16]. We used Am-utils 6.1b3: it contains over 60,000 lines of C code in 430 files. The build process begins by running several hundred small configuration tests to detect system features. Then it builds a shared library, ten binaries, four scripts, and documentation: a total of 152 new files and 19 new directories. Although the Am-utils compile is CPU intensive, it contains a fair mix of file system operations, which result in the creation of several files and random read and write operations on them. This compile benchmark was done for Ext2, as well as for I^{3}FS for the aforementioned six configurations.

For an I/O-intensive benchmark we used Postmark [23], a popular file system benchmarking tool. Postmark creates a large number of files and continuously performs operations that change the contents of the files to simulate a large mail server workload. We configured Postmark to create 20,000 files (between 512 bytes and 10KB) and perform 200,000 transactions in 200 directories. Postmark was run on Ext2 and I^{3}FS with NP, MDI, MWI, and MPI configurations. The other configurations, MD, MW, and MP, are not relevant for Postmark, as Postmark creates a lot of new files and these configurations only apply to existing files.

Finally, to measure the performance of I^{3}FS with frequency of checks, we wrote a custom program that repeatedly performs read operations on a single file. We conducted this test for I^{3}FS with frequency of checks set to 1, 2, 4, 8, 16, and 32.

Am-utils Results

Figure 4 shows the overheads of I³FS under different configurations for an Am-utils compile.

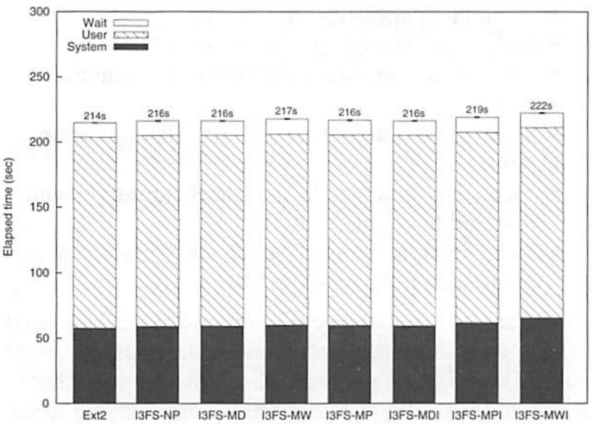


Figure 4: Am-utils results for Ext2 and I³FS.

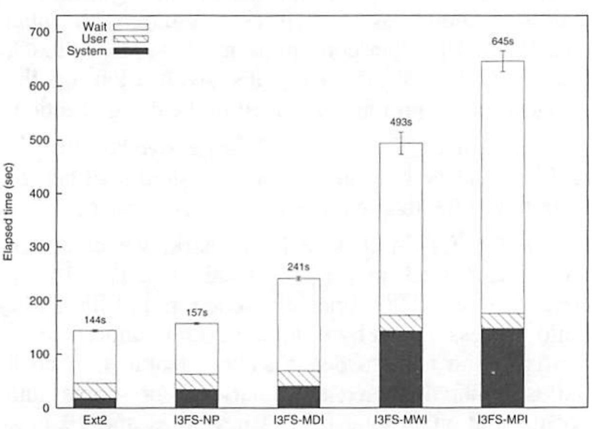


Figure 5: Postmark results for Ext2 and I³FS.

The configuration of I³FS that has the maximum overhead when compared to regular Ext2 is the MWI configuration. It has an overhead of 4% elapsed time and 13% system time. Most of the elapsed time overhead is due to system time increase because of the checksum computation. The MWI configuration calculates data checksums for whole files including the files that are newly created. Therefore, it has the highest overhead of all configurations. The elapsed time overheads of all other configurations are less than 1%. The MPI configuration has a system time overhead of 7% as this configuration computes data checksums for files including newly created ones. The system time overhead of other configurations range from 2% to 3%.

Since an Am-utils compile represents a normal user workload, we conclude that I³FS performs reasonably well under normal conditions.

Postmark Results

Figure 5 shows the overheads of Ext2 and I³FS for Postmark under the NP, MDI, MPI, and configurations. Since Postmark creates and accesses files on its own, it can only exercise configurations that have inheritable policies.

Unlike the Am-utils compile, for Postmark we were able to see a wide range of overheads for different configurations of I³FS. The NP configuration had an elapsed time overhead of 9%. The system time overhead was 89%, which is mainly because of the check for the existence of policies. The overhead due to indirection of the stackable layer also adds to this overhead. The MDI configuration had an elapsed time overhead of 67%. This overhead is partly because of checksum computation for the meta data during file creation and accessing. Database operations for storing and retrieving meta data checksums also contribute to the overall overhead. I³FS under the MWI configuration was 3.5 times slower than Ext2. This is because it computes, stores, and retrieves checksums for the data and meta data of all files, including newly created files. Finally, the MPI configuration was 4.5 times slower than Ext2. The MPI configuration checksums the meta data and the individual pages of all the files. The MPI configuration of I³FS is slower than the MWI configuration because we configured Postmark to create files whose sizes range from 512 bytes to 10KB. Thus the maximum number of pages a file can have is three as the page size is 4KB, and computing the checksums for the three pages in one shot is more efficient than checksumming individual pages separately.

Since Postmark creates 20,000 files and performs 200,000 transactions within a short period of just 10 minutes, it generates a rather intensive I/O workload. In normal multi-user systems, such workloads are unlikely. The above benchmark shows a worst case performance of I³FS. Under normal conditions, the overheads of I³FS are reasonably good, as evident from the Am-utils compile results shown later.

Frequency of Checks

To measure the performance of I³FS for whole file checksumming with frequency of checks enabled, we wrote a custom user level program that reads the first page of a 64 MB file 500 times. We ran this test with 64 MB RAM, so as to ensure that cached pages are flushed when the file is read sequentially during checksum computation phase. Since the file is read sequentially, by the time the last page of the 64 MB file is read, we can be sure that the first page is flushed out of memory. We calculated the difference in speeds for frequency values of 1, 2, 4, 8, 16, and 32. These numbers reduce the frequency that the checksums are computed logarithmically. Figure 6 shows the results of our custom benchmark for the different values of frequency of checks.

As evident from the figure, the time taken is reduced logarithmically as the frequency number increases exponentially. We can see that the rate of decrease of the elapsed time and the system time is almost equal. This is because both the I/O for reading

the files and the checksum computation itself reduces as the frequency value increases. Without a custom frequency value (Freq-1), the program takes 1,393 seconds to complete, and as the frequency value increases, the time taken reduces to 464 seconds, 279 seconds, and so on.

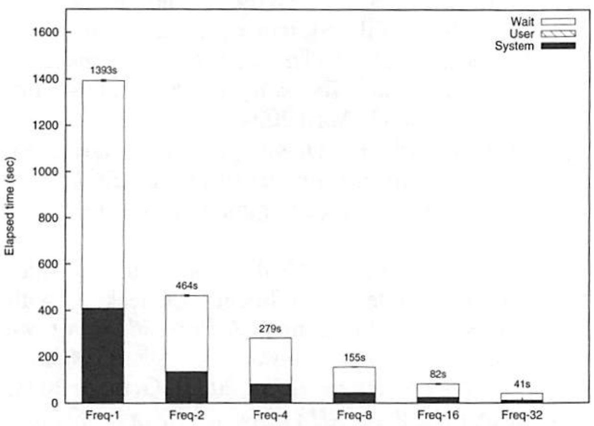


Figure 6: Frequency-of-checks benchmark results for frequency values of 1, 2, 4, 8, 16, and 32.

Therefore, when system administrators are concerned about the system performance while checksumming whole files, they can set an appropriate frequency of check value.

Related Work

In recent years, systems researchers have proposed various alternatives to increase the security of computer systems. These solutions can be broadly categorized as user mode and kernel mode. Integrity checking of files is an important aspect of system security. Our work is an in-kernel approach to check integrity and detect intrusions in the file system.

In this section we briefly discuss some previous work that addresses integrity checking and file system security in a broader sense. We discuss this in three categories: user-mode utilities, in-kernel approaches and other approaches that increase the security of the computer systems.

User Tools and Utilities The open-source community has developed various user-mode tools for file system integrity checking. While we follow the semantics of Tripwire [10, 9, 22] for our integrity checker, there are other similar tools. These include Samhain [20], Osiris [15], AIDE [18], and Radmind [3]. Most of these user-mode tools were modeled along the lines of Tripwire. AIDE and Radmind have been developed for UNIX systems with some more functions like threaded daemons and easy system management. In addition, Samhain uses a stealth mode of intrusion detection with remote administration utilities.

In-Kernel Approach *Linux Intrusion Detection System* (LIDS) [12] is a more comprehensive system that modifies the Linux access control semantics, called discretionary access control (DAC), thus offering mandatory access control inside the kernel. In contrast, our work does not change any Linux semantics or any modifications to the kernel. We leverage file system level call interposition using a loadable kernel module for file systems.

A similar approach is used by *Linux Security Modules* (LSM) [24], which present an extensible framework with in-kernel hooks for adding new security mechanisms inside the kernel for file systems, memory management and the network sub-systems. LSM does not use any policies, but provides a foundation to add complete system security.

Other Systems Security Apart from integrity checkers, there are other ways to increase the security of any host. System call interposition [17] uses the indirection of any call to a kernel function. This is a powerful tool for monitoring application behavior as soon as the context switches to kernel mode. Ostia [6] presents a model that delegates certain system critical responsibilities to a sandbox. This helps in localizing the impact of any attack after a pseudo off-line detection process. In contrast, our approach uses interposition of calls made by the virtual file system (VFS) on behalf of a file system.

Another class of solutions uses call-graph analysis to backtrack any intrusions on a host [11]. These techniques aim to determine the vulnerability of the system used by the attacker to break into the system after an attack took place. In contrast, I^{3}FS tries to detect an intrusion or inconsistency in the system as it occurs.

Finally, more recent work uses Virtual Machine Monitors (VMM) to detect any intrusions by placing the IDS in a more secure hardware domain [6]. This approach aims to minimize the impact of an attacker on the intrusion detection system. This approach has been tested for passive attack scenarios and incurs system overhead due to context switches across the interface between the OS and the VMM. In contrast, our approach has less overhead since we use fine-grained indirection.

Conclusions

We have described the design, operation, security, and performance of a versatile integrity checking file system. A number of different policy options are provided with various levels of granularity. System administrators can customize I^{3}FS with the appropriate options and policies so as to get the best use of it, keeping in mind performance requirements. As evident from the benchmark results, I^{3}FS has a performance overhead of 4% compared to regular Ext2 under normal user workloads. The encrypted database and cryptographic checksums make I^{3}FS a highly secure and reliable system.

Future Work

Our group has previously developed secure and versatile file systems like NCryptfs [25, 27], Tracefs

[1] and Versionfs [13] and we would like to integrate the features of these file systems together with I^{3}FS so as to provide a highly secure and versatile system.

Currently, I^{3}FS cannot be customized to individual users. We plan to add per user policies and options, so that individual users can set up security options for their own files, without requiring the intervention of the system administrators, but still allow administrators to override global policies.

Acknowledgments

We thank the anonymous Usenix LISA reviewers for the valuable feedback they provided, and our shepherd Michael Gilfix. Thanks go to Charles P. Wright for his suggestions; this work was based on integrity protection ideas and experience developed by Charles within the NCryptfs encryption file system. This work was partially made possible by an NSF CAREER award CNS-0133589, NSF Trusted Computing award CCR-0310493, and HP/Intel gift numbers 87128 and 88415.1.

The open-source software described in this paper is available in http://www.fsl.cs.sunysb.edu/project-i3fs.html .

Author Information

Swapnil Patil (swapnil@cs.sunysb.edu) completed his Master's degree in Computer Science in Summer 2004 from Stony Brook University. He is interested in systems and networking research with a focus on resilient distributed systems, wireless networks (sensor, 802.11), and autonomic Internet-scale systems. At present, Swapnil is working with NEC Labs America in Princeton, NJ.

Anand Kashyap (anand@cs.stonybrook.edu) completed his Bachelor's degree in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, India. He is currently pursuing his Ph.D. in Computer Science at Stony Brook University. He is working in the Wireless Networking and Simulation Lab under the guidance of Prof. Samir Das. His research interests include wireless networking and systems security.

Gopalan Sivathanu (gopalan@cs.stonybrook.edu) obtained his Bachelor's degree in Computer Science and Engineering from the University of Madras, Chennai, India. He is currently pursuing his Ph.D. in Computer Science at Stony Brook University. He is working in the Filesystems and Storage Lab under the guidance of Prof. Erez Zadok. His research interests include operating systems, systems security, and distributed computing.

Erez Zadok (ezk@cs.stonybrook.edu) is on the Computer Science faculty at Stony Brook University, the author of "Linux NFS and Automounter Administration" (Sybex, 2001), the creator and maintainer of the FiST stackable templates system, and the primary maintainer of the Am-utils (a.k.a. Amd) automounter. Erez conducts operating systems research with a focus on file systems, security, versatility, and portability.

Bibliography

[1] Aranya, A., C. P. Wright, and E. Zadok, "Tracefs: A File System to Trace Them All," *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pp. 129-143, March/April 2004.

[2] CERT Coordination Center, *CERT/CC Overview incident and Vulnerability Trends Technical Report,* http://www.cert.org/present/cert-overview-trends .

[3] Craig, W. and P. McNeal, "Radmind: The Integration of Filesystem Integrity Checking with File System Management," *Proceedings of the 17th USENIX Large Installation System Administration Conference (LISA 2003)*, October 2003.

[4] DesAutels, P. A., *SHA1: Secure Hash Algorithm,* http://www.w3.org/PICS/DSig/SHA1_1_0.html , 1997.

[5] Deutsch, P., *Independent Implementation of MD5 (RFC 1321)*, http://www.opensource.apple.com/darwinsource/10.2.3/cups-30/cups .

[6] Garfinkel, T. and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of the Network and Distributed System Security Symposium*, February 2003.

[7] Ghemawat, S., H. Gobioff, and S.-T. Leung, "The Google File System," *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29-43, October 2003.

[8] Kashyap, A., J. Dave, M. Zubair, C. P. Wright, and E. Zadok, *Using Berkeley Database in the Linux kernel,* http://www.fsl.cs.sunysb.edu/project-kbdb.html , 2004.

[9] Kim, G. and E. Spafford, "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection," *Proceedings of the Usenix System Administration, Networking and Security (SANS III)*, 1994.

[10] Kim, G. and E. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," *Proceedings of the Second ACM Conference on Computer Communications and Society (CCS)*, November 1994.

[11] King, S. and P. Chen, "Backtracking Intrusions," *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, October 2003.

[12] LIDS Project, *Linux Intrusion Detection System,* http://www.lids.org .

[13] Muniswamy-Reddy, K., C. P. Wright, A. Himmer, and E. Zadok, "A Versatile and User-Oriented Versioning File System," *Proceedings of*

the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pp. 115-128, March/April 2004.

[14] Nechvatal, J., E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, *Report on the Development of the Advanced Encryption Standard (AES)*, Technical report, Department of Commerce: National Institute of Standards and Technology, October 2000.

[15] Osiris, *Osiris: Host Integrity Management Tool*, http://www.osiris.com.

[16] Pendry, J. S., N. Williams, and E. Zadok, *Am-utils User Manual*, 6.1b3 edition, http;//www.am-utils.org, July 2003.

[17] Provos, N., "Improving Host Security with System Call Policies," *Proceedings of the 12th Annual USENIX Security Symposium*, August 2003.

[18] Reed, J., *File Integrity Checking with AIDE*, http://www.ifokr.org/bri/presentations/aide_gslug-2003/, 2003.

[19] Rivest, R. L., "RFC 1321: The MD5 Message-Digest Algorithm," *Internet Activities Board*, April 1992.

[20] Samhain Labs, *Samhain: File System Integrity Checker,* http://samhain.sourceforge.net.

[21] Seltzer, M. and O. Yigit, "A New Hashing Package for UNIX," *Proceedings of the Winter USENIX Technical Conference*, pp. 173-84, http://www.sleepycat.com, January 1991.

[22] Tripwire Inc, *Tripwire Software,* http://www.tripwire.com.

[23] VERITAS Software, *VERITAS File Server Edition Performance Brief: A PostMark 1.11 Benchmark Comparison*, Technical Report, Veritas Software Corporation, http://eval.veritas.com/webfiles/docs/fsedition-postmark.pdf, June 1999.

[24] Wright, C., C. Cowan, J. Morris, S. Smalley, and G. Kroah Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[25] Wright, C. P., M. Martino, and E. Zadok, "Cryptfs: A Secure and Convenient Cryptographic File System," *Proceedings of the Annual USENIX Technical Conference*, pp. 197-210, June 2003.

[26] Zadok, E. and I. Badulescu, "A Stackable File System Interface for Linux," *LinuxExpo Conference Proceedings*, pages 141-151, May 1999.

[27] Zadok, E., I. Badulescu, and A. Shender, "Cryptfs: A Stackable Vnode Level Encryption File System," Technical Report CUCS-021-98, Computer Science Department, Columbia University, http://www.cs.columbia.edu/library, June 1998.

[28] Zadok, E. and J. Nieh, "FiST: A Language for Stackable File Systems," *Proceedings of the Annual USENIX Technical Conference*, pp. 55-70, June 2000.

Nix: A Safe and Policy-Free System for Software Deployment

Eelco Dolstra, Merijn de Jonge, and Eelco Visser – Utrecht University

ABSTRACT

Existing systems for software deployment are neither safe nor sufficiently flexible. Primary safety issues are the inability to enforce reliable specification of component dependencies, and the lack of support for multiple versions or variants of a component. This renders deployment operations such as upgrading or deleting components dangerous and unpredictable. A deployment system must also be flexible (i.e., policy-free) enough to support both centralised and local package management, and to allow a variety of mechanisms for transferring components. In this paper we present Nix, a deployment system that addresses these issues through a simple technique of using cryptographic hashes to compute unique paths for component instances.

Introduction

Software deployment is the act of transferring software to the environment where it is to be used. This is a deceivingly hard problem: a number of requirements make effective software deployment difficult in practice, as most current systems fail to be sufficiently *safe* and *flexible*.

The main safety issue that a software deployment system must address is *consistency*: no deployment action should bring the set of installed software components into an inconsistent state. For instance, an installed component should never be able to refer to any component not present in the system; and upgrading or removing components should not break other components or running programs [15], e.g., by overwriting the files of those components. In particular, it should be possible to have multiple versions and variants of a component installed at the same time. No duplicate components should be installed: if two components have a shared dependency, that dependency should be stored exactly once.

Deployment systems must be flexible. They should support both *centralised* and *local package management*: it should be possible for both site administrators and local users to install applications, for instance, to be able to use different versions and variants of components. Finally, it must not be difficult to support deployment both in source and binary form, or to define a variety of mechanisms for transferring components. In other words, a deployment system should provide flexible *mechanisms*, not rigid *policies*.

Despite much research in this area, proper solutions have not yet been found. For instance, a summary of twelve years of research in this field indicates, amongst others, that many existing tools ignore the problem of interference between components and that end-user customisation has only been slightly examined [6]. Consequently, there are still many hard outstanding deployment problems (see the first section), and there seems to be no general deployment system available that satisfies all the above requirements. Most existing tools only consider a small subset of these requirements and ignore the others.

In this paper we present Nix, a safe and flexible deployment system providing mechanisms that can be used to define a great variety of deployment policies. The primary features of Nix are:

- Concurrent installation of multiple versions and variants
- Atomic upgrades and downgrades
- Multiple user environments
- Safe dependencies
- Complete deployment
- Transparent binary deployment as an optimisation of source deployment
- Safe garbage collection
- Multi-level package management (i.e., different levels of centralised and local package management)
- Portability

These features follow from the fairly simple technique of using cryptographic hashes to compute unique paths for component instances.

Motivation

In this section we take a close look at the issues that a system for software deployment must be able to deal with.

Dependencies For safe software deployment, it is essential that the *dependencies* of a component are correctly identified. For correct deployment of a component, it is necessary not only to install the component itself, but also all components which it may need. If the identification of dependencies is incomplete, then the component may or may not work, depending on whether the omitted dependencies are already present on the target system. In this case, deployment is said to be *incomplete*.

As a running example for this paper we will use the *Subversion* version management system (http://subversion.tigris.org/). It has several (optional) dependencies, such as on the Berkeley DB database library. When we package the Subversion component for deployment, we must take this into account and ensure that the Berkeley DB component is also present on each target system. But it is easy to forget this! This is because such dependencies are often picked up "silently." For instance, Subversion's configure script will detect and use Berkeley DB automatically if present on the build system. If it is present on the target system, Subversion will happen to work; but if it is not, it won't: an incomplete deployment. Some existing deployment systems use various tricks to automate dependency identification, e.g., RPM [11] can use the ldd tool at packaging time to scan for shared library dependencies. However, such approaches are either not general enough or not portable.

Variability Components may exist in many *variants*. Variants occur when different versions exist (i.e., almost always), and when a component has optional features that can be selected at build time. This is known as variability [24]. The Subversion component has several optional features, such as whether we want support for OpenSSL encryption and authentication, whether only a Subversion client should be built, and whether an Apache server module should be built so that Subversion can act as a WebDAV server. Of course, there also exist many different versions of Subversion, which we sometimes want to use in parallel (for instance, to test a new version before promoting it to production use on a server). A flexible deployment system should support the presence of multiple variants of a component on the same system. For instance, on a multi-user system different users may have different requirements and therefore need different variants; on a server system we may want to test a new component before upgrading critical server software to use it; or other components may have conflicting requirements on some component.

Consistency Unfortunately, most package management disciplines do not support variants very well. Deployment operations (such as installing, upgrading, or renaming a component) are typically *destructive*: files are copied to certain locations within the file system, possibly overwriting what was already there. This can destroy the consistency among components: if we upgrade or delete some component, then another component that depends on it may cease to work properly. Also, it makes it hard to have multiple variants of a component installed concurrently, that is, different versions of the component, or a version built with different parameters. For instance, the RPM packages for Subversion contain files such as /usr/bin/svn, making it impossible to have two versions installed at the same time. Worse, we might encounter unsatisfiable requirements, e.g., if two applications both require mutually incompatible versions of some library.

Atomicity Component upgrades in conventional systems are not *atomic*. That is, while a component is being overwritten with a newer version, the component is in an inconsistent state and may well not work correctly. This lack of atomicity extends beyond the level of individual components. When upgrading an entire system, for instance, it may be necessary to upgrade shared components such as shared libraries first. If they are not backwards compatible, then there will be a timing window in which components that use them fail to work properly.

Identification Variants make identification of dependencies surprisingly hard. We may say that a component depends on glibc-2.3.2, but what are the exact semantics of such a statement? For instance, it does not identify the build parameters with which glibc has been built, nor is there any guarantee that the identifier glibc-2.3.2 always refers to the same entity in all circumstances. Indeed, versions of Red Hat Linux and SuSE Linux both have RPM packages called glibc-2.3.2, but these are not the same, not even at the source level (they have vendor-specific patches applied).

Source/binary deployment We must often create both "source" and "binary" packages for a component. Creating the latter manually is unfortunate, since binary deployment can be considered an optimisation of source deployment because it uses fewer resources on the target system. Ideally, the creation of binary packages would happen automatically and transparently, but in practice, the creation and dissemination of binary packages requires explicit effort. This is particularly the case if multiple variants are required (which variants do we build, and how do users select them?).

The source/binary dichotomy complicates dependency specification, since a component can have different dependencies at build time and at run time that must be carefully identified. This is tricky, since a build time dependency can become a run time dependency if the construction process *stores* a reference to its dependencies in the build result – a *retained dependency*. For instance, various libraries such as OpenSSL are inputs to the Subversion build process. If they are shared libraries, then their full paths (e.g., /usr/lib/libssl.so.0.9.6) will be stored in the resulting Subversion executables, causing these build time dependencies to become run time dependencies. However, if they are *statically* linked (which is a build time option of Subversion), then this does not occur. Thus, there is a subtle interaction between variant selection and dependencies.

Centralised vs. local package management To make software deployment efficient, system administrators should not have to install each and every application separately on every computer on a network. Rather, software installation should be managed centrally. On the other hand, computers or individual users may have individual software requirements. This

requires local package management. Software deployment should cater for both local and centralised package management. It should not be hard to define machine-local policies.

Overview

The Nix software deployment system is designed to overcome the problems of deployment described in the previous section. The main ingredients of the Nix[1] system are the *Nix store* for storing isolated installations of components; *user environments*, providing a user view of a selection of components in the store; *Nix expressions*, specifying the construction of a component from its sources; and a generic means of *sharing* build results between machines. These ingredients provide *mechanisms* for implementing a wide variety of deployment *policies*. In this section we give a high-level overview of these ingredients from the perspective of users of the system. In the next section their implementation is described.

Nix Store

The fundamental problem of current approaches to software deployment is the confusion of *user space* and *installation space*. An end-user interacts with the applications installed on a computer through a certain interface. This may be the *start menu* on Windows and other desktop environments, or the PATH environment variable in command-line interfaces on Unix-like systems. These interfaces form what we call the *user space*. Deployment is concerned with making applications available through such interfaces by installing all files necessary for their operation in the file system, i.e., in the *installation space*.

Mainly due to historical reasons – deployment was often done manually – user space and installation space are commonly identified. For instance, to keep the list of directories in the PATH manageable, applications are installed in a few fixed locations such as /usr/bin. Thus, management of the end-user interface to applications is equal to physical manipulation of installation space, entailing all the problems discussed in the previous section.

In Nix, user space and installation space are separated. User space is a *view* of installation space. Applications and all programs and libraries used to implement them are installed in the Nix *store*. Each component is installed in a separate directory in the store. Directory names in the store are chosen so as to uniquely identify revisions and variants of components. This identification scheme goes beyond simple name+version schemes, since these cannot cope with variants of the same version of a component. Thus, multiple versions of a component can coexist in the store without interference.

[1]The name *Nix* is derived from the Dutch word *niks*, meaning *nothing*; build actions do not see anything that has not been explicitly declared as an input.

Nix Expressions

Installation of components in the store is driven by *Nix expressions*. These are declarative specifications that describe all aspects of the construction of a component, i.e., obtaining the sources of the component, building it from those sources, the components on which it depends, and the constraints imposed on those dependencies. Rather than having specific built-in language constructs for these notions, the language of Nix expressions is a simple functional language for computing with *sets of attributes*. Figure 1 shows a Nix function that returns variants of the Subversion system, based on certain parameters; it features most typical constructs of the language. Figure 2 shows a call to this function. We will use these examples to explain the elements of the language.

```
{ clientOnly, apacheModule, sslSupport
, stdenv, fetchurl, openssl, httpd
, db4 }:

assert !clientOnly -> db4 != null;
assert apacheModule -> !clientOnly;
assert sslSupport -> (openssl != null
  && (apacheModule ->
       httpd.openssl == openssl));

derivation {
  name = "subversion-0.32.1";
  system = stdenv.system;

  builder = ./builder.sh;
  src = fetchurl {
    url =
      http://.../subversion-0.32.1.tgz;
    md5 = "b06717a8ef50db4b...";
  };

  # Pass these to the builder.
  inherit clientOnly apacheModule
          sslSupport;
          stdenv openssl httpd db4;
}
```

Figure 1: Subversion component (subversion.nix).

```
stdenv = import ...;
openssl = import ...;
... # other component definitions

subversion = (import subversion.nix) {
  clientOnly = false;
  apacheModule = false;
  sslSupport = true;
  inherit stdenv fetchurl openssl
          httpd db4 expat;
};
```

Figure 2: Subversion composition (pkgs.nix).

Derivation The body of the expression is formed by calling the primitive function derivation with an *attribute set* {key=value;...}. The set contains two attributes required by the derivation function: the builder attribute indicates a script that builds the component, while the system attribute specifies the target platform

on which the build is to be performed. The other attributes define values for use in the build process (such as dependencies) and are passed to the build script as environment variables. The name attribute is a symbolic identifier for use in the high-level user interface; it does not necessarily uniquely identify the component.

Parameters In order to describe variants of a component, an expression can be *parameterised*, i.e., turned into a *function* from Nix expressions to Nix expressions. The syntax for functions is $\{k_1, ..., k_n\}$: body, which defines a function that expects to be called with an attribute set containing attributes with names k_1 to k_n. Thus, the Subversion expression is parameterised with expressions describing the components on which it depends (e.g., openssl, httpd, stdenv), options that select features (e.g., clientOnly, sslSupport), and a utility (fetchurl). The stdenv component provides all the basic tools that one would expect in a Unix-like environment, e.g., a C compiler, linker, and standard Unix utilities. Parameters are instantiated in a function application. For example, the expression in Figure 2 instantiates the Subversion expression by assigning values to its parameters.

A subtle but important difference with most component formalisms is that in Nix we explicitly describe not just components but also compositions of components. For instance, an RPM spec file specifies how to build a component, but not its dependencies. It merely states fairly weak conditions on the expected build environment ("a package called glibc-2.3.2 should be present"). Thus, a spec file is always incomplete, so there is no way to uniquely specify concrete components. The Subversion Nix expression in Figure 1 is similarly incomplete, but the composition in Figure 2 provides the whole picture – information on how to build not just Subversion, but also all of its dependencies.

The value of the src attribute is another example of functional computation. Its value is the result of a call to the function fetchurl (passed in as an argument of the Subversion function) that downloads the source from a specific URL and verifies that it has the right MD5 checksum.

Assertions In order to restrict the values that can be passed as parameters, a function can state assertions over the parameters. For example, the db4 database is needed only when a local server is implemented. Also, *consistency* between components can be enforced. For instance, if both SSL and Apache support are enabled, then Apache must link against the same OpenSSL library as Subversion, since at runtime the Subversion code will be linked against Apache. If this were not enforced, link errors could result.

Build When a derivation is built, the build script indicated by the builder attribute is invoked. As stated above, attributes of the derivation are passed through environment variables to the builder. In the case of attributes that refer to other derivations (i.e., dependencies), the corresponding environment variables contain the paths at which they are stored. Nix ensures that such dependencies are built prior to the invocation of the builder, so the build script can assume that they are present. The special variable out conveys to the builder where it should store the build result. Figure 3 shows the build script for Subversion. The largest part of the script is used to compute the configuration flags based on the features selected for the Subversion instance. By using a user-definable script for implementing the build of a component, rather than building in a specific build sequence, no requirements have to be made on the build interface of source distributions.

```
buildInputs="$openssl $db4 $httpd"
# Bring in GCC etc., set up environment.
. $stdenv/setup

if ! test $clientOnly; then
  extraFlags="--with-berkeley-db=$db4 \
  $extraFlags"
fi

if test $sslSupport; then
  extraFlags="--with-ssl \
    --with-libs=$openssl $extraFlags"
fi

...
tar xvfz $src
cd subversion-*
./configure --prefix=$out $extraFlags
make
make install
```

Figure 3: Subversion build script (builder.sh).

User Environments

A Nix *user environment* consists of a selection of applications from the store currently relevant to a user. "Users" can be human users, but also system users such as daemons and servers that need a specific selection to be visible. This selection may be implemented in various ways, depending on the interface used by the user. In the case of the PATH interface, a user environment is implemented as a single directory – the counterpart of /usr/bin – containing symbolic links (or wrapper scripts on systems that do not support them) to the selected applications. Thus, manipulation of the user environment consists of manipulation of this collection of symbolic links, rather than directories in the store. Installation of an application in user space entails adding a symbolic link to a file in the store and uninstallation entails removing this symbolic link instead of physically removing the corresponding file from the file system.

While other approaches (e.g., [4]) also use a directory with symbolic links, these are composed manually and/or are only provided in a single location. In Nix an environment is a component in the store. Thus, any number of environments can coexist and variant environments can be composed with tools.

This separation of user space and installation space allows the realization of many different deployment scenarios. The following are some typical examples:

- A user environment may be prescribed by a system administrator, or may be adapted by individual users.
- Different users on the same system can compose different user environments, or can share a common environment.
- A single user can maintain multiple 'profiles' for use in different working situations.
- A user can experiment with a new version of a component while keeping the old (stable) version around for regular tasks.
- Upgrading to a new version or rolling back to an old one is a matter of switching environments.
- Removal of unused applications can be achieved by automatic *garbage collection*, taking the applications in user environments as roots.

For instance, to add the Subversion component in Figure 2 to the current user environment, we do:

```
$ nix-env -f pkgs.nix -i subversion
```

where pkgs.nix is the file containing the definition in Figure 2. This will build Subversion and create a new user environment, based on the old one, to which Subversion has been added. If an expression for a new Subversion release comes along, we can upgrade as follows:

```
$ nix-env -f pkgs.nix -u subversion
```

which likewise creates a new user environment, based on the old one, in which the old Subversion component has been replaced by the new one. However, the old user environment and the components included in it are retained, so it is possible to return to the old situation if necessary:

```
$ nix-env --rollback
```

There is no operation to physically remove components from the system. They can only be removed from a user environment, e.g.,

```
$ nix-env -e subversion
```

creates a new user environment from which the links to Subversion have been removed. However, storage space can be reclaimed by periodically running a garbage collector:

```
$ nix-collect-garbage
```

which removes any component not reachable from any user environment. (Therefore it is necessary to periodically prune old user environments, e.g., once we find that we do not need to roll back to old ones). Garbage collection is safe because we know the full dependency graph between components.

Sharing Component Builds

The unique identification of a component in the store is based on all the inputs to the build process, thus capturing all special configurations of the particular variant being built. Thus, components can be identified exactly and deterministically. Consequently a component can be shared by all components that depend on it. Indeed we even get *maximal sharing*: if two components are the same, then they will occupy the same location in the store. This means that builds can be shared by users on the same machine.

Since the identification only depends on the inputs to the build process and the location of the store, store identifiers are even *globally unique*. That is, a component build can be safely copied to a Nix store on another machine. For this purpose, Nix provides support for transparently maintaining a collection of pre-built components on some shared medium such as an FTP site or an installation CD-ROM. After building a component in the store it can be *pushed* to the shared medium.

For instance, the installation and upgrade operations above perform an installation from source. This is generally not desirable since it is slow. However, it is possible to safely and transparently re-use pre-built components from a shared resource such as a network repository. For instance, a component distributor or system administrator can pre-build components, then *push* (upload) them to a server using PUT requests:

```
$ nix-push http://example.org/cache \
                        pkgs.nix subversion
```

This will build Subversion (if necessary) and upload it and all its dependencies to the indicated site. A user can then make Nix aware of these:

```
$ nix-pull http://example.org/cache
```

Subsequent invocations of nix-env -i / -u will automatically use these *if* they are exactly equal to what the user is requesting to be installed. That is, if the user changes any sources, flags, and so on, the pre-built components will *not* be used, and Nix will revert to building the components itself. Thus, Nix is both a source and binary-based deployment system; deployment of binaries is achieved transparently, as an optimisation of a source-based deployment process.

Policies

Nix is *policy-free*. That is, the ingredients introduced above are *mechanisms* for implementing software deployment. A wide variety of *policies* can be based on these mechanisms.

For instance, depending on the type of organisation it may or it may not be desirable or possible that users install applications. In an organisation where homogeneity of workspaces is important, the selection and installation of applications can be restricted to system administration. This can be achieved by restricting all the operations on the store, and the composition of user environments to system administration. They may compose several prefab user environments for different classes of users. On the other hand, for instance in

a research environment, where individual users have very specific needs, it is desirable that users are capable of installing and upgrading applications themselves. In this situation environment operations and the underlying store operations can be made available to ordinary users as well. Similarly, Nix enables deployment at different levels of granularity, from a single machine, a cluster of machines in a local network, to a large number of machines on separate sites.

Many other policies are possible; some are discussed later.

Implementation

In this section we discuss the implementation of the Nix system. We provide an overview of the main components of the system, which we then discuss in detail.

The Store

The two main design goals of the Nix system are to *support concurrent variants*, and to *ensure complete dependency information* which is necessary to support *completeness* (the deployment process should transmit all dependencies necessary for the correct operation of a component). It turns out that the solutions to these goals are closely related. Other design goals are portability (we should not fundamentally rely on operating system specific features or extensions) and storage efficiency (identical components should not be stored more than once).

The first problem is dealing with variability, i.e., concurrent variants. As we hinted in the previous section, we support this by storing each variant of a component in a global *store*, where they have unique names and are isolated from each other. For instance, one version or variant of Subversion might be stored in /nix/store/eeeeaf42e56b-subversion-0.32.1,[2] while another might end up in /nix/store/3c7c39a10ef3-subversion-0.34. To ensure uniqueness, these names are computed *by hashing all inputs involved in building the component.*

Thus, each object in the store has a unique name, so that variants can co-exist. These names are called *store paths*. In Autoconf [1] terminology, each component has a unique prefix. The file system content referenced by a store path is called a *store object*. Note that a given store path uniquely determines the store object. This is because two store objects can only differ if the inputs to the derivations that built them differ, in which case the store path would also differ due to the hashing scheme used to compute it. Also, a store object can never be changed after it has been built.

Figure 4 shows a number of derivates in the store. The tree structure simply denotes the directory hierarchy. The arrows denote dependencies, i.e., that the file at the start of the arrow contains the path name of the file at the end of the arrow, e.g., the program svn depends on the library libc.so.6, because it lists the file /nix/store/8d013ea878d0-glibc-2.3.2/lib/libc.so.6 as one of the shared libraries against which it links at runtime.

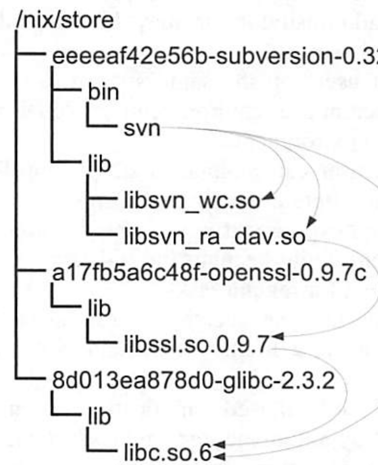


Figure 4: The Store.

The use of these names also provides a solution for the dependency problem. First, it prevents undeclared dependencies. While it is easy for hard-coded paths (such as /usr/bin/perl) to end up in component source, thereby causing a dependency that is easily forgotten while preparing for deployment, no developer would manually write down these paths in the source (indeed, being the hash of all build inputs, they are much too "fragile" to be included). Second, we can now actually *scan* for dependencies. For instance, if the string 3c7c39... appears in a component, we know that it has a dependency on a specific variant of Subversion 0.34. This in particular solves the problem of retained dependencies (discussed in the first section): it is not necessary to declare explicitly those build time dependencies that, through retention, become run time dependencies, since we can find them automatically.

With precise dependency information, we can achieve the goal of complete deployment. The idea is to always deploy *component closures*: if we deploy a component, then we must also deploy its dependencies, their dependencies, and so on. That is, we must always deploy a set of components that is closed under the "depends on" relation. Since closures are self-contained, they are the units of complete software deployment. After all, if a set of components is *not* closed, it is not safe to deploy, since using them might cause other components to be referenced that are missing on the target system.

Building Components

So how do we build components from Nix expressions? This could be expressed directly in terms of Nix expressions, but there are several reasons why this is a bad idea. First, the language of Nix expressions is fairly high-level, and as the primary interface for

[2]The actual names use 32 hexadecimal digits (from a 128-bit cryptographic hash), but they have been shortened here to preserve space.

developers, subject to evolution; i.e., the language changes to accommodate new features. However, this means that we would have to be able to deal with variability in the Nix expression language itself: several versions of the language would need to be able to coexist in the store. Second, the richness of the language is nice for users but complicates the sorts of operations that we want to perform (e.g., building and deployment). Third, Nix expressions cannot easily be identified uniquely. Since Nix expressions can import other expressions scattered all over the file system, it is not so straightforward to generate an identifier (such as a cryptographic hash) that uniquely identifies the expression. Finally, a monolithic architecture makes it hard to use different component specification formalisms on top of the Nix system (e.g., we could retarget Makefiles to use Nix as a backend).

For these reasons Nix expressions are translated into the much simpler language of *store expressions*, just as compilers generally do the bulk of their work on simpler intermediate representations of the code being compiled, rather than on a full-blown language with all its complexities. Store expressions describe how to build one or more store paths. *Realisation* of a store expressions means making sure that all those paths are present in the store.

Derivation store expressions describe the building of a single store component. They describe all inputs to the build process: other store expressions that must be realised first (build time dependencies), the build platform, the build script (which is one of the dependencies), and environment variable bindings. These are computed from calls to the derivation function in the Nix expression language by recursively translating all input derivations to derivation store expressions, copying source files to the store, and adding all attributes as environment variable bindings.

To perform the build action described by a derivation, the following steps are taken:

1. Locks are acquired on the output path (the store path of the component being built) to ensure correctness in case of parallel invocations of Nix.
2. Input store expressions are realised. This ensures that all file system inputs are present.
3. The environment is cleared and initialised to the bindings specified in the derivation.
4. The builder is executed.
5. If the builder was executed successfully, we build a *closure store expression* that describes the resulting closure, i.e., the output path and all store paths directly or indirectly referenced by it. We do this by scanning every file in the output path for occurrences of the cryptographic hashes in the input store paths. For instance, when we build Subversion, the path /nix/store/a17fb5a...-openssl-0.9.7c is passed as an input. After the build, we find that the string a17fb5a... occurs in the file libsvn_ra_dav.so (as shown in Figure 4). Thus, we find that Subversion has a retained dependency on OpenSSL. Build time dependencies carried over to runtime are detected automatically in this way. (This approach is discussed in more detail in [9]).
6. The closure expression is written to the store.

The command nix-instantiate translates a Nix expression to a store expression:

```
$ nix-instantiate pkgs.nix
/nix/store/ce87...-subversion.store
```

The command nix-store --realise realises a derivation store expression, returning the resulting closure store expression:

```
$ nix-store --realise \
  /nix/store/ce87...-subversion.store
/nix/store/ab1f...77ef.store
```

Nix users do not generally have to deal with store expressions. For instance, the nix-env command hides them entirely – the user interacts only with high-level Nix expressions, which is really just a fancy wrapper around the two commands above. However, store expressions are important when implementing deployment policies. Their relevance is that they give us a way to uniquely identify a component both in source and binary form, through the derivation and closure store expression, respectively. This can be used to implement a variety of deployment policies.

A crucial operation for deployment is to query the set of store paths referenced by a store expression. This is the set of paths that must be copied to another system to ensure that it can realised there. For instance, for the derivation above we get:

```
$ nix-store --qR \
  /nix/store/ce87...-subversion.store
/nix/store/ce87...-subversion.store
/nix/store/d1bc...0aa1-builder.sh
/nix/store/f184...3ed7-gcc.store
/nix/store/f199...0719-bash.store
...
```

That is, this set includes the derivation store expressions for building Subversion itself and its direct and indirect dependencies, a closure store expression for the builder, and so on.

On the other hand, for the closure we get:

```
$ nix-store --qR \
     /nix/store/ab1f...77ef.store
/nix/store/ab1f...77ef.store
/nix/store/eeee...e56b-subversion-0.32.1
/nix/store/a17f...c48f-openssl-0.9.7c
/nix/store/8d01...78d0-glibc-2.3.2
...
```

This set only includes the closure store expression itself and the component store paths it references.

Substitutes

With just the mechanisms described above, Nix would be a source-based deployment system (like the

FreeBSD Ports collection [2], or Gentoo Linux [3]), since all target systems would have to do a full build of all derivations involved in a component installation. This has the advantage of flexibility. Advanced users or system administrators can adapt Nix expressions to build a variant specifically tailored to their needs. For instance, required functionality disabled by default can be enabled, unnecessary functionality can be disabled, or the components can be built with specific optimisation parameters for the target environment. The resulting derivates may be smaller, faster, easier to support (e.g., due to reduced functionality), and so on. On the other hand, the obvious disadvantages are that source-based deployment requires substantial resources on the target system, and that it is unsuitable for the deployment of closed-source products.

The Nix solution is to allow source-based deployment to change transparently into binary-based deployment through the mechanism of *substitutes*. For any store path, a *substitute expression* can be registered, which is also just a store derivation expression. Then, whenever Nix is asked to realise a closure that contains path p, and p does not yet exist, it will first try to build its substitute if available. The idea is that the substitute performs the same build as the original expression, but with fewer resources. Typically, this is done by fetching the pre-built contents of the output path of the derivation from the network, or from installation media such as a CD-ROM. This mechanism is generic (policy-free), because it does not force any specific deployment policy onto Nix. Specific policies are discussed later.

Deployment Policies

A useful aspect of Nix is that while it is conceptually a source-based deployment system, it can transparently support binary deployment through the substitute mechanism. Thus, efficient deployment consists of two aspects:

- *Source level*: Nix expressions are deployed to the target system, where they are translated to store expressions and built (e.g., through nix-env).
- *Binary level*: Pre-built derivates are made available, and substitute expressions are registered on the target system. This latter step is largely transparent to the users. There is no apparent difference between a "source" and a "binary" installation.

Source level deployment is unproblematic, since Nix expressions tend to be small. Typical deployment policies are to obtain sets of Nix expressions packaged into a single file for easier distribution, or to fetch them from a version management system. The latter is useful as it can easily allow automatic upgrades of a system. For instance, we can periodically (e.g., from a cron job) update the Nix expressions and build the derivations described by them. Note that any subexpressions that have not changed do not need to be rebuilt.

Binary level deployment presents more interesting challenges, since even small Nix expressions can, depending on the variability present in the expressions, yield an exponentially large set of possible store objects. Also, these store objects are large and may take a long time to build. Thus, we have to decide *which* variants are pre-built, *who* builds them, and *where* they are stored.

Let us first look at the most simple deployment policy: a fixed selection of variants are pre-built, *pushed* onto a HTTP server, from where they can then be *pulled* by clients. To push a derivation, all elements in the resulting closure are packaged (e.g., by placing them into a .tar.gz archive). All of this is entirely automatic: to push the derivations of some expression foo.nix the distributor merely has to issue the command nix-push foo.nix.

The client issues the command nix-pull to obtain a list of available pre-built components available from a pre-configured URL (i.e., the HTTP server). For each derivation available on the server, substitute expressions are registered that (when built) will fetch, decompress, and unpack the packaged output path from the server. Note that nix-pull is *lazy*: it will not fetch the packages themselves, just some information about them.

```
subversion = {apacheModule, stdenv}:
  (import ./subversion.nix)
  { clientOnly = false
  , sslSupport = true
  , apacheModule = apacheModule
  , stdenv = stdenv, ... };

subversion' = {stdenv}:
  [(subversion {apacheModule = true})
   (subversion {apacheModule = false})];

subversion'' =
  [(subversion'
       {stdenv = stdenv-Linux})
   (subversion'
       {stdenv = stdenv-FreeBSD})];
```

Figure 5: Variant selection.

The issue of *which* variants to pre-build requires the distributor to determine the set of variants that are most likely to be useful. For instance, for the Subversion component, it may never be useful to *not* have SSL support, but it may certainly be useful to leave out Apache server support, since that feature introduces a dependency on Apache, which might be undesirable (e.g., due to space concerns). Also, the platform for which to build must be selected. Figure 10 shows how four variants of Subversion can be built. The function subversion supplies all arguments of the expression in Figure 1, except apacheModule and stdenv (which determines the build tools, and thus the target platform). The function subversion' uses this to produce two variants, given a stdenv: one with Apache server support,

and one without. This function is in turn used by the variable subversion," which calls it twice, with a stdenv for Linux and FreeBSD respectively. Hence, this evaluates to $2 \times 2 = 4$ variants.

Pre-building and pushing to a shared network site merely optimises deployment of common variant selections; it does not preclude the use of variants that are not pre-built. If a user selects a variant for which no substitute exists, the variant will be built locally from source. Also, input components such as compilers that are exclusively build time dependencies (that is, they appear in the derivation value but not in the closure value) will only be fetched or built when the variant must be built locally.

The tools nix-pull and nix-push are not part of the Nix system as such; they are applications of the underlying technology. Indeed, they are just short Perl scripts, and can easily be adapted to support different deployment policies. For instance, an entirely different policy is lazy construction, where clients push derivates onto a server if they are not already present there. This is useful if it is not known in advance which derivates will be needed. An example is mass installation of components in a heterogeneous network. In a peer-to-peer architecture each client makes its derivates available to all other clients (that is, it pushes onto itself, and pulls from all other clients). In this case there is no server, and thus, no need to provide central storage scaling in the number of clients.

User Environment Policies

The use of cryptographic hashes in store paths gives us reliable identification of dependencies and non-interference between components, but we can hardly expect users to type, e.g., /nix/store/eeeeaf42e56b-subversion-0.32.1/bin/svn when they want to start a program! Clearly, we should hide these implementation details from users.

We solve this problem by *synthesising user environments*. A user environment is the set of applications or programs available to the user through normal interaction mechanisms, which in a Unix setting means that they appear in a directory in the user's PATH environment variable. The user has in her PATH variable the path /nix/links/current/bin. /nix/links/current is a symbolic link (symlink) that points to the current user environment *generation*. Generations are symlinks to the actual user environment. They are needed to implement atomic upgrades and rollbacks: when a derivation is added or removed through nix-env, we build the new environment, and then create a generation symlink to it with a number one higher than the previous generation. User environments are just sets of symlinks to programs of activated components (similar to, e.g., GNU Stow [4]), and are themselves computed using derivations.

This is illustrated in Figure 6 (dotted lines denote symlinks), where the current symlink points to generation 42, which is in turn a symlink to a user environment in the store. The user environment is simply a tree of symlinks to activated components. Hence, the path /nix/links/current/bin/svn indirectly refers to /nix/eeee...-subversion-0.31.1/bin/svn.

Figure 6 also shows what happens when we upgrade Subversion, and add Mozilla in a single atomic action. A new environment is constructed in the store based on the current generation (42), the new generation (43) is made to point to it, and finally the current link is switched to point at generation 43. The semantics of the POSIX rename() system call ensures that this is an atomic operation. That is, users and programs always see the old set of activated programs, or the new set, but never neither, both, or a mix. Since old generations are retained, we can atomically downgrade to them in the same manner.

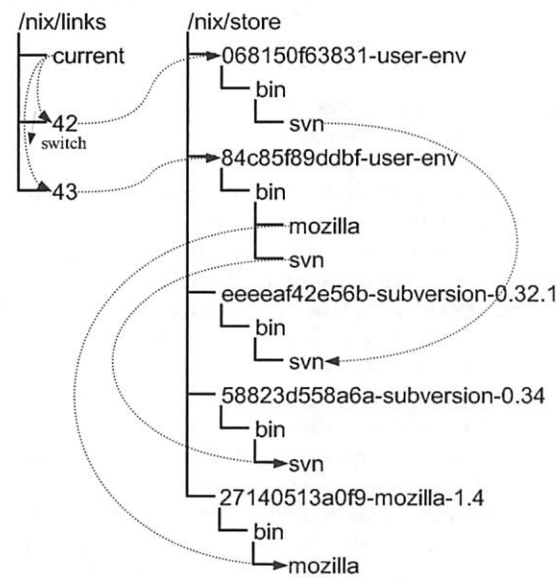


Figure 6: User environments.

The generation links are the only external links into the store. This means that the only reachable store paths are those in the closure of the targets of the generation links. The closure can be found using the closure values computed earlier. Since all store paths not in this closure are unreachable, they can be *deleted* at will. This allows Nix to do automatic *garbage collection* of installed components. Nix has no explicit operation to delete a store path – that would be unsafe, since it breaks the integrity of closures containing that path. Rather, it provides operations to remove derivations from the user environment, and to garbage collect unreachable store paths. Store paths reachable only from old generations can be garbage collected by removing the generation links.

This scheme, where a user environment is created for the entire system, is just the simplest user environment policy. The creation of a user environment is itself a normal derivation, and the command

nix-env used in the second section is a simple wrapper that automatically creates a derivation, builds it, and switches the current generation to the resulting output path. The build script used by nix-env for environment creation is a fairly trivial Perl script that creates symlinks to the files in its input closures. A simple modification is to allow *profiles* – environments for specific users or situations. This can be done by specifying a different link directory (e.g., /home/joe/.nixlinks). Also, multiple versions of the same program in an environment can be accommodated through renaming (e.g., a symlink svn-0.34), which is a policy decision that can be implemented by modifying the user environment build script.

```
derivation {
  name = "site-env";
  builder = ./create-symlinks.pl;
  inputs = [
    ((import ./subversion.nix) { ... })
    ((import ./mozilla.nix) { ... })
    ... ];
}
```

Figure 7: A Nix expression to build a site-wide user environment (site-wide.nix).

A more interesting extension is *stacked* user environments, where one environment links to the programs in another environment. This is easily accommodated: just as the inputs to the construction of an environment can be concrete components (such as Subversion), they can be other environments. The result is another indirection in the chain of symlinks. A typical scenario is a 2-level scheme consisting of a site-wide environment specified by the site system administrators, with user-specific environments that augment or override the site-wide environment. Concretely, the site administrator makes a Nix expression as in Figure 7 (slightly simplified) and makes it available on the local network. Locally, a user can then link this site-wide environment into her own environment by doing

```
nix-env -f site-wide.nix -i site-env
```

where site-wide.nix refers to the Nix expression. This will replace any previously installed derivation with the symbolic name site-env. To ensure that changes to the site-wide environment are automatically propagated, these commands can be run periodically (e.g., from a cron job), or initiated centrally (by having the administrator remotely execute them on every machine and/or for every user).

Should components in the local environment override those in the site-wide environment? Again, this is a policy decision, and either possibility is just a matter of adapting the builder for the local user environment, for instance to give precedence to derivations called site-env.

Server configurations User environments (contrary to what the term implies) can not only be used to specify environments for specific users, but also for specific tasks or processes. In particular, they can be used to specify complete *server configurations*, which includes not only the software components constituting some server, but also its configuration and other auxiliary files. Consider, for instance, an Apache/Subversion server (the Subversion server runs as a module on top of Apache). It consists of several components that are rather picky about specific dependencies, e.g., Apache, Subversion, ViewCVS, Python, and Perl, but also our repository management CGI scripts, static HTML documents and images, the Apache httpd.conf configuration file, SSL private keys, and so on. Since these are also components (just not necessarily executable components) they can be managed using Nix.

Figure 8 shows a (simplified) Nix expression for an Apache/Subversion server. It takes a single argument that specifies whether a test or production server is to be built. The builder produces a component consisting of an Apache configuration file, and a control script to start and stop the server. The builder generates these by substituting values such as the desired port number and the paths to the Apache and Subversion components into the given source files.

```
{productionServer}:
derivation {
  builder = ./builder.sh;
  configuration = ./httpd.conf.in;
  controller = ./ctl.sh.in;
  portNumber = if productionServer
               then 80 else 8080;
  inherit (import ...) httpd subversion;
}
```

Figure 8: A Nix expression to build a Subversion server.

Now, given a simple script upgrade-server (not shown here) that uses nix-env -u to build the new server configuration, stop the server running in the old generation, and start the new one, we can easily instantiate new server configurations by editing source files such as httpd.conf.in, and calling upgrade-server. For instance, the command upgrade-server test instantiates the Nix expression by calling it with a false argument, thus producing a test server. If this is found to work properly, we can issue upgrade-server production to upgrade the production server. nix-env --rollback can be used to go back to the previous generation, if necessary.

The server is started using the script controller.sh which is part of the server configuration component. It initialises PATH to point to a specific set of components. This means that the server configuration is self-contained: it does not depend on anything not explicitly specified in the Nix expression. Such a configuration is therefore pretty much immune to external configuration changes, and can be relatively easily transferred to another machine.

The only thing not under Nix control here is *state* – things that are modified by the server, e.g., the actual Subversion repositories and user account databases.

Thus, Nix can be used for the deployment of not just software components, but also complete system configurations – the domain of tools such as Cfengine [7]. Note that Cfengine declaratively specifies *destructive changes* to be performed to realise a desired configuration. This makes it hard to easily run several configurations in parallel on the same machine, or to switch back and forth between configurations. Also, Cfengine is typically not used to manage the software components on a machine (although this is possible, e.g., by installing the appropriate packages in an Cfengine action [20]).

Experience

We have applied Nix to a number of problem domains.

Software deployment We have "nixified" 180 or so existing Unix packages, including large ones such as Mozilla Firefox with all its dependencies (which includes the C compiler, basic Unix tools, X11, etc.). They are prebuilt for Linux and made available through the push/pull mechanism.

The fundamental limitation to Nix's dependency checking is that it will not prevent undeclared dependencies on components outside of the store. For instance, if a builder calls /bin/sh, we have no way to detect this. To minimise the probability of such undeclared dependencies, we use patched versions of gcc, ld, and glibc that refuse to use header files and libraries outside of the Nix store. In our experience this works quite well. For instance, the prebuilt Nix packages work on a variety of Linux distributions – evidence that no (major) external components are used. A common problem with these distributions is that they often differ in subtle ways that cause packages built on one system to fail on another, e.g., because of C library incompatibilities. However, our Nix components are completely boot-strapped, that is, they are built using only build tools, libraries, etc., that have themselves been built using Nix, and do not rely on components outside of the Nix store (other than the running kernel). Using our reliable dependency analysis, any required libraries and other components are deployed also. Thus, they just "work."

The ability to very rapidly perform rollbacks is often a life-saver. For instance, it happens quite frequently that we attempt to upgrade some bleeding-edge software package, only to discover that it doesn't work quite as well as the previous version (or not at all!). A simple nix-env --rollback saves the day. In most package managers, recovery would be much harder, since we would have to know exactly what the previous configuration was, and we would have to have a way to re-obtain the old versions of the packages that were just upgraded.

Service deployment As described later, Nix can be used for the deployment of not just software components, but also complete configurations of system services. For instance, our department's Subversion server is managed in this way. The main advantages are that it is very easy to run multiple instances of a service (e.g., for testing – and the test server will in no way interfere with the production server!), that it is easy to move a service to another machine since we have full dependency information, and again that we can rollback to earlier versions.

Build farms It is a good software engineering practice to build software systems continuously during the development process [13]. In addition, if software is to be portable, it should be built on a variety of machines and configurations. This requires a *build farm* – a set of machines that sit in a loop building the latest version obtained from the version management system. Build farms are also important for *release management* – the production of software releases – which must be an automatic process to ensure reproducibility of releases, which is in turn important for software maintenance and support.

The management of a build farm is often highly time-consuming. For instance, if the component being built in the build farm requires (say) Automake 1.7, we must install that version of Automake on each machine in the build farm. If at some point we need a newer version of Automake, we again must go to each machine to perform the upgrade. So maintaining a build farm scales badly. Worse, there may be conflicting dependencies (e.g., some other component in the build farm may only work with Automake 1.6).

Such management of dependencies is exactly what Nix is good at, so we have implemented a build farm on top of Nix. The main advantages over other build farms (e.g., [12]) are:

- The Nix expression language makes it easy to describe the build tasks, along with their dependencies.
- Nix ensures that the dependencies are installed on each machine in the build farm.
- The hashing scheme ensures that identical builds (e.g., of dependencies) are performed only once.
- In Nix, each derivation has a system attribute that specifies on what kind of platform the derivation is to be performed (e.g., i686-linux). If the attribute does not match the type of the platform on which Nix is run, Nix can automatically distribute the derivation to a different machine of the intended platform type, if one exists. All inputs to the derivation are copied to the store of the remote machine, Nix is run on the remote machine, and the result is copied back to the local store. Thus, dealing with multi-platform builds is fairly transparent: we can write a Nix expression specifying derivations on a variety of platforms and run it on a arbitrary machine. There is no need to schedule the build separately on each machine.

- The resulting builds can be used immediately by other developers since they are made available through nix-push.

A downside to a Nix-based build farm is that installing a package through Nix differs from the "native" way of installing a package on existing platforms (e.g., by installing an RPM on a Red Hat machine). Thus it is difficult for a Nix build farm to verify whether a package works when built from source in the native way. However, on Linux systems, we can in fact build native packages (such as RPMs) without affecting the host system by using User-Mode Linux [5] in Nix derivations. In fact, this fits in quite well. For instance, the synthesis of the UML disk images for the various platforms for which we build packages is just a normal Nix derivation that creates an Ext2 file system from an arbitrary set of RPMs constituting a Linux distribution.

Related Work

Centralised and local package management Package management should be centralised but each machine must be adaptable to specific needs [26]. Local package management is often ignored in favour of centralised package management [19, 17]. In our approach, central configurations can easily be shared and local additions can be made. Any user can be allowed to deviate from a central configuration. Software installation by arbitrary users is discussed in [21]. In [25] policies are introduced that define which installation tasks are permitted. This might be a challenging extension to Nix. Modules [14] makes software deployment more transparent by abstracting from the details of software deployment. Application-specific deployment details are captured in "modulefiles," which can be shared between large-scale distributed networks, similar to Nix expressions. Modules lack the safety properties of Nix. As a result, correct operation of typical deployment tasks, as discussed initially, cannot be guaranteed.

Non-interference Software packages should not interfere with each other. Typical interference is caused by attempting to have multiple versions of a component installed. It is important that multiple versions can coexist [21], but this is difficult to achieve with current technology [6]. A common approach is to install software packages in separate directories, sometimes called *collections* [26]. In [18], a directory naming scheme is used that restricts the number of concurrent versions and variants of a package. Sharing is in most deployment systems either unsafe due to implicit references, or not supported at all because every application is made completely self-contained [17, 19]. Sharing of data across platforms using a directory structure that separates platform specific from platform independent data is discussed in [17], which is concerned with diversity in platform, not diversity in feature sets. As a consequence however, exchange and sharing of packages is not truly safe, as is the case for Nix.

Safe upgrading Many systems ignore this issue [26]. Automatic rollback on failures is discussed in [19]. This turned out to be undesirable in practice because it increased installation time and did not increase consistency. RPM [11] has a notion of transactions: if the installation of a set of packages failed, the entire installation is undone. This is not atomic, so the packages being upgraded are in an inconsistent state during the upgrade. The approach discussed in [18] uses shortcuts to default package versions, e.g., emacs pointing to emacs-20.2. This is unsafe because programs may now use emacs which initially corresponds to emacs-20.2, but after an upgrade points to, e.g., emacs-20.3. Separation of production and development software via directories is discussed in [17]. Once an application has been fully tested under the development tree it is turned into production. This requires recompilation because path names will change and may cause errors. Consequently, the approach is not really safe.

Garbage collection In [21] an approach for removing old software is discussed. Basically, after software is "removed" by making the directory unreadable, one verifies whether other software fails by running it. If so, the deletion is rolled back by making the directory readable again. This is unsafe because the test executions may not reveal every dependency, and because a time window is introduced during which some components do not work.

Dependency analysis However, the same paper also describes a pointer scanning mechanism similar to ours: component directories are scanned for the names of other component directories (e.g., tk-3.3). However, such names are not very unique (contrary to cryptographic hashes) and may lead to many false positives. Also, component dependencies are scanned for *after* the component has been made unreadable, not before. In [23] a dependency analysis tool for dynamic libraries is discussed. In Nix this information is already available when an application is installed, and Nix is not restricted to detecting dependencies on shared libraries only. Vesta [16] is a system for configuration management that supports automatic dependency detection. Like Nix, it detects only dependencies that are actually needed, and dependencies are complete, i.e., every aspect of the computing environment is described and controlled by Vesta.

Safety In [25] common *wrong* assumptions of package managers are explained, including: i) package installation steps always operate correctly; ii) all software system configuration updates are the result of package installation. In Nix, software gets installed safely, without affecting the environment. Thus, in contrast to many other systems, Nix will never bring a system in an unstable state. Unless a system administrator

really wants to mess things up, all upgrades to the Nix store are the result of package installation. Safe testing of applications outside production environments is discussed in [21, 17]. In [15] it is confirmed that software should be installed in private locations to prevent interference between software packages. Interference turns out to be a very common cause of installation problems. In Nix, such packages can safely coexist.

Packaging In [22] a generic packaging tool for building bundles (i.e., collections of products that may be installed as a unit) is discussed. Source tree composition [8] is an alternative technique for automatically producing bundles from source code components. However, these bundling approaches do not cater for sharing of components *across* bundles.

Conclusion

Seemingly simple tasks such as installing or upgrading an application often turn out to be much harder than they should be. Unexpected failures and the inability to perform certain actions affect users of all levels of computing expertise. In this paper we have pinpointed a number of causes of the deployment malady, and described the Nix system that addresses these by using cryptographic hashes to enforce uniqueness and isolation between components. It is successfully used to deploy software components to several different operating systems, to manage server configurations, and to support a build farm.

There are a number of interesting issues remaining. Of particular interest is our expectation that Nix will permit *sharing* of derivations between users. That is, if user A has built some derivation, and user B attempts to build the same derivation, B can transparently reuse A's result. Clearly, using code built by others is not safe in general, since A may have tampered with the result. However, our use of cryptographic hashes can make this safe, since the hash includes all build inputs, and therefore completely characterises the result.

The problems of dependency identification and dealing with variants also plague build managers such as Make [10]. We believe that (with some extensions) Nix can be used to replace these more low-level software configuration management tools as well.

Availability

Nix is free software and is available online at http://www.cs.uu.nl/groups/ST/Trace/Nix .

Acknowledgements

We wish to thank Martin Bravenboer and Armijn Hemel for helping in the development of the Nix system, and Martin Bravenboer and our LISA shepherd Rudi van Drunen for commenting on this paper. This research was supported by CIBIT|Serc and the NWO Jacquard program.

Author Information

Eelco Dolstra is a Ph.D. student in the Software Technology group at Utrecht University, where he also obtained his Master's degree on the integration of functional and strategic term-rewriting languages. His research focuses on dealing with variability in software systems and configuration management, in particular software deployment.

Merijn de Jonge obtained his Ph.D. degree from the University of Amsterdam in the area of software reuse. After a postdoc position at Eindhoven University, he currently works as a postdoc at Utrecht University. His research interests include software reuse, configuration and build management, software variability, generative programming, component-based software development, program transformation, and language-centered software engineering.

Eelco Visser studied computer science at the University of Amsterdam where he obtained Master's and Ph.D. degrees in the area of syntax definition and language processing. As a postdoc at the Oregon Graduate Institute in Portland he laid the foundation for the Stratego program transformation language. He is currently an assistant professor in the Software Technology group at Utrecht University where he leads research projects into program transformation and software configuration and deployment.

References

[1] *Autoconf*, http://www.gnu.org/software/autoconf/ .

[2] *FreeBSD Ports Collection*, http://www.freebsd.org/ports/ .

[3] *Gentoo Linux*, http://www.gentoo.org/ .

[4] *GNU Stow*, http://www.gnu.org/software/stow/ .

[5] *User Mode Linux*, http://user-mode-linux.sourceforge.net/ .

[6] Anderson, E. and D. Patterson, "A Retrospective on Twelve Years of LISA Proceedings," *Proceedings of the 13th Systems Administration Conference (LISA '99)*, pp. 95-107, November 1999.

[7] Burgess, Mark, "Cfengine: A Site Configuration Engine," *USENIX Computing systems*, 8(3), 1995.

[8] de Jonge, Merijn, "Source Tree Composition," *Seventh International Conference on Software Reuse*, Num. 2319, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[9] Dolstra, E., E. Visser, and M. de Jonge, "Imposing a Memory Management Discipline on Software Deployment," *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 583-592, IEEE Computer Society, May 2004.

[10] Feldman, Stuart I., "Make – A Program for Maintaining Computer Programs," *Software –*

Practice and Experience, Vol. 9, Num. 4, pp. 255-265, 1979.

[11] Foster-Johnson, Eric, *Red Hat RPM Guide*, John Wiley and Sons, 2003.

[12] Mozilla Foundation, *Tinderbox*, http://www.mozilla.org/tinderbox.html .

[13] Fowler, Martin, *Continuous Integration*, http://www.martinfowler.com/articles/ continuousIntegration.html .

[14] Furlani, J. L. and P. W. Osel, "Abstract Yourself with Modules," *Proceedings of the Tenth Systems Administration Conference (LISA '96)*, pp. 193-204, September 1996.

[15] Hart, John and Jeffrey D'Amelia, "An Analysis of RPM Validation Drift," *Proceedings of the 16th Systems Administration Conference (LISA '02)*, pp. 155-166, USENIX Association, November 2002.

[16] Heydon, Allan, Roy Levin, Timothy Mann, and Yuan Yu, *The Vesta Approach to Software Configuration Management*, Technical Report Research Report 168, Compaq Systems Research Center, March 2001.

[17] Manheimer, K., B. A. Warsaw, S. N. Clark, and W. Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform Boundaries," *Proceedings of the Fourth Systems Administration Conference (LISA '90)*, pp. 37-46, October 1990.

[18] Oetiker, T., "SEPP: Software Installation and Sharing System," *Proceedings of the 12th Systems Administration Conference (LISA '98)*, pages 253-259, December 1998.

[19] Oppenheim, K. and P. McCormick, "Deployme: Tellme's Package Management and Deployment System," *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, pages 187-196, December 2000.

[20] Ressman, D. and J. Valdés, "Use of Cfengine for Automated, Multi-platform Software and Patch Distribution," *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, pp. 207-218, December 2000.

[21] Rouillard, J. P. and R. B. Martin, "Depot-lite: A Mechanism for Managing Software," *Proceedings of the Eighth Systems Administration Conference (LISA '94)*, pages 83-91, 1994.

[22] Staelin, C., "mkpkg: A Software Packaging Tool," *Proceedings of the 12&th Systems Administration Conference (LISA '98)*, pp. 243-252, December 1998.

[23] Sun, Y. and A. L. Couch, "Global Impact Analysis of Dynamic Library Dependencies," *Proceedings of the 15th Systems Administration Conference (LISA 2001)*, pp. 145-150, November 2001.

[24] van Gurp, Jilles, Jan Bosch, and Mikael Svahnberg, "On the Notion of Variability in Software Product Lines," *Proceedings of WICSA 2001*, August 2001.

[25] Venkatakrishnan, V., N. R. Sekar, T. Kamat, S. Tsipa, and Z. Liang, "An Approach for Secure Software Installation," *Proceedings of the 16th Systems Administration Conference (LISA '02)*, USENIX Association, pp. 219-226, November 2002.

[26] Wong, W. C., "Local Disk Depot: Customizing the Software Environment," *Proceedings of the Seventh Systems Administration Conference (LISA '93)*, pages 49-53, November 1993.

Auto-configuration by File Construction: Configuration Management with Newfig

William LeFebvre and David Snyder – CNN Internet Technologies

ABSTRACT

A tool is described that provides for the automatic configuration of systems from a single description. The tool, newfig, uses two simple concepts to provide its functionality: boolean logic for making decisions and file construction for generating the files. Newfig relies heavily on external scripts for anything beyond the construction of files. This simple yet powerful design provides a mechanism that can easily build on other tools rather than a single monolithic stand-alone program. This provides for a great deal of flexibility while maintaining simplicity. The language is a combination of boolean logic and output statements, and also provides for macros and other essential elements. All output is written to channels: an abstraction which provides for extensive configurability. Examples are provided that show the tool's power and flexibility. A description is also provided of the efforts undergone at CNN to integrate this tool in to a sizable infrastructure. The paper concludes with a discussion of future improvements.

Introduction

As the number of systems in an infrastructure grows, the administration problem grows with it. Unless the engineering staff grows accordingly, at some point the management of the systems will need to be automated. This realization is nothing new, and several successful tools have been developed to meet this need.

Configuration data used by a system to control how it behaves can be broken down into three basic categories: information which is unique to a system, information which is common to a proper subset of systems, and information which is common to all systems in the infrastructure. Unique information includes a system's hostname, local disk partitions, and network address assignments. Global information includes such files as */etc/services* and */etc/networks*. The partially global information is the most interesting: it is common across systems which share a similar function but not necessarily others. The task of configuring a system requires that all of this information be in place and correct. The most complicating factor in achieving a correct configuration is in the management of files which combine data from more than one category.

A file that only contains unique data can be copied into place when a system is installed. Such files are rarely touched after system installation. Files that contain only global information can be easily updated from a central repository. Likewise, files that contain partially global information can be updated from a central authority provided there is some mechanism that distinguishes among the different classes of systems and is able to determine which data are appropriate for the system. However, when a file contains a mixture of these data categories, its maintenance becomes significantly more difficult. This is most noticeable in files such as *fstab*, *inetd.conf*, *hosts.allow*, rc script directories, and sometimes *passwd*.

When we sought out a tool for automated systems configuration, we were looking for certain properties that we believe would be beneficial in our environment. We wanted the system to be idempotent, congruent, deterministic, transportable, extensible, and fail-safe. We wanted a specification language that was both clear and concise, to minimize training and maximize understanding. When a tool could not be found that met all of these criteria, we set out to design a new solution to the problem. The result, *newfig*, takes a new approach to the problem while providing all of the qualities that we believe are important. As a result, very little is built in to *newfig*. Instead it is a framework which can utilize other tools and scripts to accomplish results. Rather than provide a wide range of built-in mechanisms, *newfig* uses file construction as its only primitive operation. Its configuration is a purely declarative language based on boolean logic. The files that *newfig* constructs, called *channels*, can be used to replace existing files on the system or as input to external programs (including scripts). As a result, the system is naturally extensible. *Newfig* is used to generate input for and monitor the execution of the programs and scripts which perform the actual modifications to the system. It provides a structure around which system administrators can do what they do best: automate through scripting. We believe that the resulting system meets all our initial design goals and provides us with an excellent platform for automated configuration.

Related Work

Prescriptions [8] is a declarative language for describing the desired state of configuration for distributed systems. It provides mechanisms for specifying operations that may be used to bring systems into conformance with a specification. However, it does

not seem to provide mechanisms for file distribution and synchronization. We have not been able to find recent work on this project since Thornton's 1994 technical report.

CFengine [1] is the most widely known work in this area. It is to automated systems configuration what awk(1) is to scripting. The main concepts are patterns, actions, and an execution context that is managed by an interpreter. It provides a rich body of built-in actions, but has little room for expansion beyond that set. The configuration drives actions to perform on a system, whereas *newfig* provides a description of the desired target for the system and enforces conformity to that description. CFengine implements convergence by providing a mechanism that brings a system closer to an ideal. In our environment we sought a tool that implements congruence, which is a tighter standard than convergence. The idea of file construction is not central to CFengine, and a CFengine configuration that uses such a methodology tends to be cumbersome. CFengine is also not able to remove changes implicitly: such steps must be explicitly given in the configuration.

Psgconf [6] takes a highly modular approach to configuration management, providing hooks for various data stores, policy rules, and actions. The configuration parsing is order dependent, making psgconf a procedural configuration tool rather than declarative, which has some advantages and some drawbacks.

Site [3] uses declarative statements to describe the configuration of a computing site at three levels of abstraction. At the lowest levels, drivers written in C are used to construct the contents of configuration files. The paper describes a prototype implementation only. In contrast, *newfig* provides no restrictions on the language used to construct channels, which is good for admins who have long since shed their systems programming scales (or never had them to begin with).

PIKT [5] is an interpreted scripting language, preprocessor, and scheduler that is primarily intended to monitor systems, reporting problems and taking corrective action when possible. Over time, it has been extended to include configuration management features, but most of the terminology in the language is built around monitoring. For example, scheduling periodic execution of a script involves adding it to the "alarms" section of the alerts.cfg file.

Radmind [2] takes a file based approach to configuration management by integrating intrusion detection with centralized system management. An advantage is that complex, out-of-band changes can be captured and incorporated into the configuration, but maintaining consistency of configuration data that is duplicated across multiple files may present a challenge without factoring tools.

ISconf [9] is a highly order dependent configuration tool based on make(1) files. A description of changes is provided to the tool, and it ensures that those changes are carried out on each system in an exact order. ISconf version 2 requires that the description be created and maintained manually, whereas later versions provide more automated ways for generating the description. The basic premise of ISconf is that "order matters": it is easier to replicate the order in which operations are conducted than it is to determine and accommodate the interactions of those operations. Like newfig, ISconf implements congruence. The difficulty with ISconf is the monotonic increase to the description and the steps required to recreate a system. As changes are piled on top of changes, the time required to build or rebuild a system continues to increase. Although preservation of the order of changes is sufficient to achieve congruence, it is our belief that it is not necessary.

Design

Newfig is a system designed to provide for the automatic configuration of individual machines from a common description. Boolean algebra is used to control the generation of output to a number of channels. Each channel can be used to control the contents and characteristics of a file. Channels can also be used as input to scripts for operations which are more complicated than basic file construction. All channel definitions are part of the configuration, allowing the functionality of *newfig* to be extended with ease. *Newfig* is designed to be idempotent, transportable, extensible, conformant (rather than convergent), and fail-safe.

The configuration consists of a series of boolean phrases interspersed with output statements. Boolean algebra is used as the logical structure for the *newfig* configuration language. Clauses are used to infer the logical value of a symbol from other symbols. The algebra supports the three basic logical operations: and, or, not. Parentheses are also recognized for grouping operations. Between the boolean clauses are statements that send lines to channels. Each channel must be explicitly defined in the configuration along with its characteristics. A channel can be associated with a file, in which case its contents becomes that of the file. A channel can also be associated with an external command or script, in which case the script is used to process the channel's contents. External commands are also used to perform syntax and semantic checks of channels' content to ensure correctness.

Processing is performed in several distinct phases in *newfig*: read, intrinsic definition, inference, macro definition, generation, filtering, instantiation. No changes are performed on the system until the instantiation step, giving *newfig* ample opportunity to discover problems before changes are made. If any problems are detected before instantiation, *newfig* will be fail-safe and not make any changes to the system.

Decisions about a system are solely dependent on the boolean clauses in the configuration; the role of

the first few phases is to evaluate the clauses. First, the entire configuration is read in and parsed. Then certain facts about the system are used to determine a set of intrinsically true symbols. The name of the system is one such symbol: it is always true. Additionally, the following symbols are intrinsically true: the name of the operating system (in all lower case), the name combined with the operating system release, and the platform type. Thus, a system named *sammy* running Solaris 9 on a SPARC platform will have the following intrinsically true symbols: *sammy*, *sunos*, *sunos-5.9*, *sparc*. Another intrinsically true symbol represents the network of the system's IP address (or addresses). For example, a system with the address 10.5.2.12 would have the symbol net.10.5.2 defined as intrinsically true. The configuration can also invoke external commands to augment the set of intrinsic symbols with either true or false values.

Once the intrinsics have been determined, the values of the other symbols in the configuration are determined through *inference*. Not every symbol's value can be determined. When inference is complete there will be a set of symbols known to be true, a set known to be false, and a set of symbols whose values are unknown. For all the remaining phases, the only symbols which matter are those known to be true.

The configuration language allows for the definition and expansion of macros in the statements and the channel declarations. In the *macro definition* phase, the values for all macros are determined. A special append operator (+=) is available to add to an existing macro definition, such as a PATH.

The *generation* phase creates the content of every channel. This is done by processing the output statements associated with true symbols.

Once the contents of each channel is known, *syntax checking* is performed by an external program as specified in the channel definition. Since *newfig* itself has no knowledge of the intended content of a channel, syntax checkers provide a way to verify that a channel's content are correct. Although this phase is named *syntax* checking, any sort of checks can be carried out by an external program. Examples of checks which could be performed are: ensure that each line of a channel has the correct number of fields, ensure a passwd channel has a line for root, verify that every program listed in *inetd.conf* exists. If any of the syntax programs indicates an error, then *newfig* will fail-safe and not make any changes to the system.

A close variant to syntax checking is the *filtering* phase. This phase is similar to syntax checking, except that the external programs invoked in this phase alter the content of the channel in addition to performing simple syntax checking. Each program acts as a Unix-style filter, reading the channel contents from standard input and writing to standard output. The results of the filter are used to replace the content of the channel. Although the order of records in most files is unimportant, it may be desirable to sort the output to improve human readability. The passwd file is commonly sorted by UID, and a filter could easily perform this task for generated passwd files. Optimization is another good use of filtering. A generated hosts.allow file may contain overlapping address ranges, and a filter could remove them as well as reformat the output to improve readability. It is important to the integrity of the entire system that filters perform no side effects and that their actions are restricted to producing output and error messages.

The final phase is *instantiation*, which is performed once all of preceding phases have executed without error. This phase ensures that the underlying system matches the configuration. For file channels, the file is compared to the generated content of the channel. If there is no difference, the file is left untouched (thus modification times are unchanged). If the channel contains different content, a new copy of the file is created and put into place. A file channel can also specify ownership and permission modes for the file. An action channel has its action command invoked with the channel contents available as standard input. Action commands usually perform side effects.

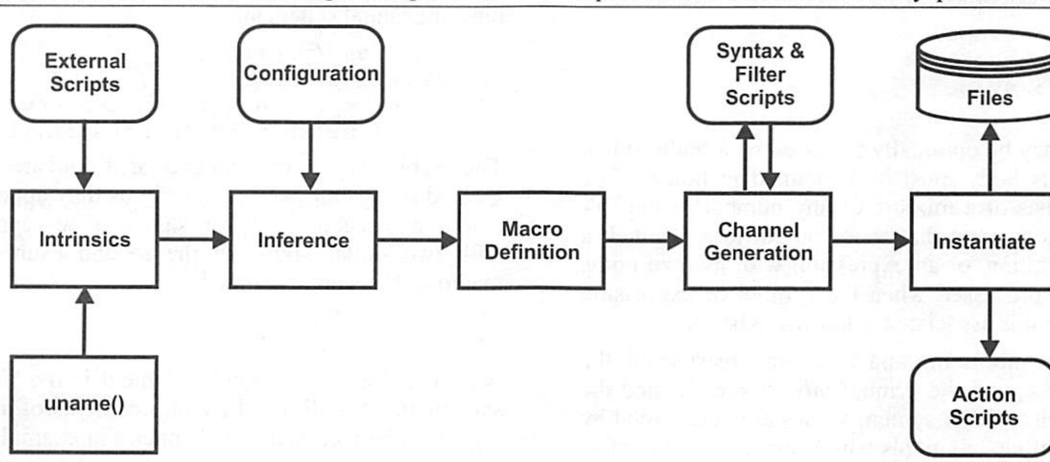


Figure 1: The phases of newfig.

A channel can have both a file and an action associated with it. In such cases, the action is only performed when the file is changed. Thus an action can be used to restart or "tickle" a daemon only after its configuration file has been updated by *newfig*. Figure 1 show the basic flow of information through the various phases.

Configuration Specifics

Each configuration statement has two separate components: a logical relationship and a body of statements. A statement need not contain both components.

Logical Clauses and Expressions

The logical relationships are specified using a simple boolean algebra. The rules for symbol names are very generous, allowing numbers, letters, and many special characters. Expressions can contain and, or, inversion, and grouping. There are two ways to specify a logical relationship: a standard clause and implication. A standard clause takes the following form:

```
symbol: expression
```

If the value of the expression can be determined, then it becomes the symbol's value. An expression consists of any combination of the following elements:

```
or      symbol1 symbol2
xor     symbol1 ^ symbol2
and     symbol1 & symbol2
not     !symbol1
```

Parentheses can be used to group expressions together, as in:

```
symbol: !(a b c d)
```

Wherever a clause can be used, an expression may also be used. An expression is just a clause with no left hand side, as in:

```
!(a b c d)
```

The second relationship is implication, where a single symbol's value is used to imply the value of a list of symbols (expressions do not make sense in this context). This takes the following form:

```
symbol -> symbol1 symbol2 symbol3 ... ;
```

This is equivalent to:

```
symbol1: symbol;
symbol2: symbol;
symbol3: symbol;
...
```

A clause may be optionally followed by a body. When present, this body must be contained in braces. The body consists of a mixture of any number of the following: a statement that sends output to a channel, a macro definition, or an expression with its own body. A body is processed when the symbol or expression with which it is associated is known to be true.

The symbols *all* and *true* are preassigned the value of true, and the symbol *false* is preassigned the value of false. Other symbol values are determined by intrinsic settings. Symbols which appear in the configuration only on the right hand side of a clause (or only on the left hand side of an implication) are called terminal symbols. The value of a terminal symbol cannot be determined by anything in the configuration. Terminal symbols which do not have an intrinsic value are assumed to be false.

Macros

Macros can be defined and expanded much like Makefile macros, although all macros are created before any statements (thus any expansion) takes place. Macros are created as follows:

```
MAC=value;
```

There is a special append operator to allow definitions to be augmented:

```
MAC+=value;
```

When expanded, such macros will have spaces separating each added component. Expansion is invoked with a dollar sign followed by the macro name in braces, for example:

```
${MAC}
```

Some macros have special meaning. The PATH macro is used as the execution path for all commands invoked by *newfig*. HOSTNAME is preset to the name of the current host. Unlike other statements, macro definition can appear outside of a clause.

Much like make(1), all macros are defined before any are expanded. Thus definition need not precede use in the configuration. The order in which macros are defined is only significant in two cases. First, if a macro is redefined then the last definition will be used. Second, the order in which macro append definitions are processed will determine the order in which the strings appear in the final macro. Currently there is no way to directly control the order in which definitions are processed other than basic sequential order.

Channel Statements

Channel statements are very simple. The statement begins with a channel name, is followed by any number of fields, and ends with a semi-colon. When processed, all fields are written to the named channel, and separated by a single space. Here are some examples of channel statements:

```
resync /opt/proftpd;
hosts.allow "ALL: 127.";
inetd.conf telnet stream tcp nowait
  root /usr/sbin/in.telnetd in.telnetd;
```

The usable characters in an unquoted field are limited. Quoted strings are written exactly as they appear after macro expansion. A dollar sign can be represented with two dollar signs. In the second example, the macro *MAC* is not expanded:

```
"string ${MAC}"
"string $${MAC}"
```

A special form of channel statement is the %include statement. This allows the entire contents of an existing file to be included in a channel. For example:

```
%include passwd /etc/base.passwds;
```

Clause and Body

The entire clause with its body must end with a semi-colon. Thus any of the following are valid clauses:

```
symbol: !(a b c d)
{
    channel line;
};
symbol: e & f;
symbol1:
{
    channel line2;
};
symbol2
{
    channel line2;
};
```

A statement or a macro definition within a body may be guarded with a boolean expression. Thus the following two clauses do the same thing:

```
x { y { channel a; }; };
x & y { channel a; };
```

The first form is more useful when the body has a number of statements, only one of which needs to be guarded, as in:

```
x {
    ch1 a;
    ch2 b;
    y { ch3 c; };
};
```

In this form, ch1 and ch2 receive output whenever x is true, but the ch3 statement is only processed when both x and y are true. This can be useful in situations where certain statements are needed only for particular operating systems.

If a clause defines the value of a symbol, then any body associated with the clause is performed whenever that symbol is true, even if it became true via a different clause. Consider the following example:

```
x: a b
{ output line1; };
x: c d
{ output line2; };
```

Note that both lines are sent to the output whenever the symbol *x* is true. More specifically, if *a* is true while *b*, *c*, and *d* are false, then the symbol *x* is asserted to be true and both lines are sent to the output, even though the second expression ("c d") is false.

Channel Definitions

There are no predefined channels. All channels must be explicitly defined in the configuration. This definition is done with a %channel clause. Following the keyword %channel is the channel name and then, in braces, its definition. For example:

```
%channel hosts {
    file /etc/hosts;
};
```

A channel may have a number of characteristics as declared in the body of the channel definition. However, a channel may only be defined once.

A channel may take on several forms, sometimes in combination. A channel that is associated with a file, or "file channel," ensures that the file contains exactly what appears in the channel. During instantiation, the contents of the channel are compared to the file, and if they differ, the file is rewritten to exactly match the channel. If they are the same, the file is left untouched. Mode and ownership are also compared and corrected where necessary.

An "action channel" has an action associated with it. The action is an external program that is run during the instantiation phase and uses the channel contents as standard input. It is expected that an action program will perform side effects.

A channel may be both an action channel and a file channel. In these cases, the action is only performed when the file is changed. This form is useful for updating daemon configuration files, as the action can ensure that the daemon is signaled or restarted.

A directory may be associated with a channel to form a "directory channel." Such a channel is expected to contain a list of files (one per line). The target directory is checked to ensure that it contains the listed files and nothing else. The contents of those files is checked and updated. Consider the following series of statements where "xinet" is a directory channel for /etc/xinetd.d:

Property	Definition
action	external action program
directory	target directory for directory channel
file	target file for file channel
filter	external program to check and augment channel contents
syntax	external program to check channel contents
owner	required file owner (for file and directory channels)
group	required file group (for file and directory channels)
mode	required permissions (for file and directory channels)
singlesource	channel contents may only come from one body
after	channel must be processed after another channel

Figure 2: Channel properties.

```
xinet /opt/proftpd/xinetd/proftpd;
xinet /opt/rsync/xinetd/rsync;
```

Each of the named files will be copied in to the target directory, and *newfig* will ensure that no other files exist in that directory.

Channel Properties

Figure 2 shows properties that can be set for each channel.

External Programs

The filter, syntax, and action properties invoke external programs, either binaries or scripts. The interface for all of these programs is the same. The contents of the channel is readable on standard input. Programs that exit with a zero status code indicate normal success, a non-zero exit code indicates that an error occurred. Lines written to standard error are considered to be error messages. If formatted correctly then *newfig* will be able to match the message up with the line that caused it. The error message format is a line number followed by a colon then the message.

Filter programs must write a revised copy of the channel contents to standard output. These results will be taken as the new channel contents. Error messages generated by filters are treated the same as the ones generated by a syntax program. Note that, in both cases, the contents of standard error is ignored unless the program exits with a non-zero status code.

In order to preserve the idempotent characteristic of *newfig*, syntax and filter programs should not produce any side effects. This includes modifying, creating, or removing files, and signaling, stopping, or starting processes. These programs can, however, use temporary files provided they are removed when the program finishes. The action programs are expected to produce side effects: that is their purpose. Although the channel contents is made available on standard input, an action program need not read it.

Interesting Properties

Newfig has a number of interesting properties that make it a strong utility. These features are generally desirable in a tool that automatically adjusts a system's configuration and behavior.

Idempotency

An operation that is *idempotent* is one that acts as if it was only invoked once even if it is invoked multiple times. An idempotent operation does not have a cumulative effect. *Newfig* is designed to be idempotent, but its ability to retain this characteristic is entirely dependent upon the action commands that it invokes.

The intrinsic symbol values for a system are entirely dependent upon characteristics of the system itself: its platform, operating system, and IP address. As long as these remain constant, the intrinsic symbols will always have the same value. However, it should be noted that the system does provide an external mechanism for augmenting the intrinsic set. If this mechanism does not provide a constant set of results, the idempotent behavior may be compromised.

As long as the intrinsic set remains constant between invocations, the results of inference will always be the same. This is insured by the boolean algebra which drives the inference phase. The set of true and false symbols derived from inference is the only means of selection for the remaining phases, guaranteeing that their results will always be the same. The only exception to this is the use of external programs for syntax, filtering, and action. Both syntax and filter commands are required to have no side effects, thus they can easily be idempotent. This leaves the action scripts. *Newfig* performs idempotent operations if and only if the action scripts it invokes also perform idempotent operations.

Transportability

The term *transportability* is being adopted to describe a characteristic of *newfig* that is unique to an automated configuration tool. A tool that is transportable is one that is capable of creating the same result regardless of the actual system on which it is run. Transportability is essential for the testability of a site's configuration. In order to perform regression tests on an infrastructure configuration, the testing mechanism must be able to determine the results of the configuration tool for a variety of systems (perhaps all) in the infrastructure. Without transportability, the generation of this data must take place on each system in the test set.

Newfig provides transportability through its strict use of boolean algebra to drive all the decisions. Assume that both the *newfig* configuration files and the external programs used by *newfig* are constant (functionally equivalent) across the infrastructure. All that *newfig* needs to be able to generate the output of each channel for a given host is the intrinsic set and the list of predefined macros for that host. *Newfig* provides mechanisms for generating just this information and using it instead of the local set. This provides transportability.

Extensibility

One of the problems with systems such as CFengine is the limited range of capability. In fairness,

```
samba {
    inetd.conf netbios-ssn stream tcp nowait root /opt/samba/bin/smbd smbd;
    inetd.conf netbios-ns dgram udp wait root /opt/samba/bin/nmbd nmbd -d2;
    resync /opt/samba;
};
alpha -> samba;
```

Figure 3: Sample Samba configuration.

CFengine has a great deal of functionality already built in, but its ability to extend that functionality is limited. *Newfig* is designed to be extensible through extensive use of external commands. These commands may be anything that can run on the native system: scripts, perl, python, pre-compiled binaries, *etc.* There is very little capability actually built in to *newfig*. The design philosophy is similar to that of the original Unix: *newfig* is a tool which can easily be used as a building block for other tools.

Conformance

The configuration for *newfig* is a complete description of the files which it controls. Rather than providing a series of steps to edit an existing file, the configuration provides enough information to reconstruct the entire file. This approach ensures that the act of removing data from a file's description will get implemented correctly on the affected machines.

Some automated configuration systems, such as CFengine, provide convergence toward an ideal. In some environments, especially those with loose control over root access, convergence is an appropriate tool for effective centralized management. However, installations which primarily consist of servers and where configuration changes are typically co-ordinated by a single organization do not need the flexibility of convergence. It is simpler to provide a complete description of a configuration and enforce conformance to it. This ensures that changes are implemented completely and with a single iteration. A complete configuration is also descriptive in what it does not contain, making the removal of unnecessary items simpler.

Consider a system that is configured to include a samba [7] server. This configuration might look like the one in Figure 3.

Such a configuration would ensure that any system for which the symbol *samba* is true would have the lines needed for samba in inetd.conf and the directory /opt/samba synced up correctly with a central repository. The last line of the configuration ensures that when *alpha* is true *samba* is true also. Thus the system *alpha* would have the smbd and nmbd lines added to inetd.conf. Other parts of the configuration would contain the remaining lines that are expected to appear in inetd.conf. Now consider what happens when the association between alpha and samba is removed, such as would be the case if it was decided that alpha should no longer provide samba service. The next time *newfig* is invoked, it will rebuild all the channels, including the inetd.conf channel. But this time it will build it without the smbd and nmbd lines. *Newfig* will detect that the resulting channel is different from the file inetd.conf and will instantiate the new channel contents as inetd.conf. Finally, if the channel definition for inetd.conf has an action command, *newfig* will run the action providing an opportunity to send a signal to inetd. The removal of this data happens as a natural consequence of the information from the configuration itself.

Fail-safe Operation

The *newfig* design defers any modifications to the system until all other processing is complete. The soundness of the configuration is checked during reading and inference. The integrity of each channel's content can be checked and augmented with syntax and filter commands. No changes are made to the system until the final phase. If any problems are found prior to instantiation, *newfig* can decide not to continue. This provides a fail-safe mechanism to ensure that an incorrect configuration is not applied to any systems.

The original design goal of *newfig* was to provide an entirely fail-safe design: any sort of errors would prevent *newfig* from instantiating any changes and executing any action. During the deployment of this 100% fail-safe model we discovered that this may not be a desirable design.

One obvious function that *newfig* can supply is driving the distribution of files from a central repository (commonly called a *gold* server). For example, a channel can contain the names of directories and files which must be kept in sync with a central server, and the action for that channel can provide the mechanism which performs the syncing. Such a channel is an obvious choice for keeping the *newfig* configuration itself in sync. Unfortunately, if *newfig* is 100% fail-safe, then any error in the configuration will completely disable this mechanism, making it impossible to recover without a mechanism outside of *newfig* itself.

Experience with a 100% fail-safe system has made it obvious that certain types of errors need not hinder the update of unrelated items. As an example, consider a configuration which maintains password files and hosts.allow files. A mistake in an entry for hosts.allow files could be something as simple as forgetting the dot at the end of a network pattern (such as "172.16.1" without the trailing dot). A properly written syntax command will catch that mistake and correctly flag it. However, a 100% fail-safe design will also prevent the passwd file from being updated, even though it has nothing to do with hosts.allow.

A better design would be to contain the failure, failing only what is affected. In the case of hosts.allow it would be contained to just its channel and none other. If the configuration states a dependency between channels, such as the *after* property, then failure of a channel should also cause failure of any channels that depend on it. *Newfig* can easily be adopted to fit this more limited idea of fail-safe.

Declarative Language

The configuration language is designed to be declarative. As a result, the order in which files are processed, and the order in which statements and clauses appear in the file do not matter. This allows

the maximum amount of flexibility for organization of the configuration information. Many of the files of concern do not depend on the ordering of lines, some important files do have this requirement. In order to accommodate this, certain guarantees are made about the order in which output statements are processed, thus affecting the order of the lines within the channels. Clauses can be processed in any order, but statements within the body of a clause will be processed in the order they appear. Line ordering in an included file will be preserved. Consider the following example:

```
x: { output a; };
x: { output b; };
```

There is no guarantee that the output will be ordered *a*, *b*. Although generally this will be true, *newfig* makes no guarantees and configurations should not rely on it. However, in the following example it is guaranteed that the lines will appear *a* followed by *b*, since both statements appear in the same body:

```
x: {
    output a;
    output b;
};
```

Examples of Practical Applications

Examples of some common applications for *newfig* are as follows.

hosts.allow

Access to services controlled by tcp wrappers [11] is set with the use of hosts.allow. This file can be controlled by *newfig*, allowing for the creation of any arbitrary hierarchy to define allowable access. The channel definition would be:

```
%channel hosts.allow {
    file /etc/hosts.allow;
    filter allow-filter;
    owner root;
    mode 444;
};
```

Although it is optional, a filter for processing this channel provides improved results. The filter can optimize the channel contents to ensure that there are no extraneous specifications in the channel. Consider the following usage of this channel:

```
all  { hosts.allow "ALL: 10.2.1.5"; };
beta { hosts.allow "ALL: 10.2.1."; };
```

For host *beta* both lines will appear in hosts.allow. A channel filter would be able to detect that the first entry is extraneous and remove it. The filter can also collapse multiple lines for the same daemon (or for ALL) in to a single line.

Services

Something as simple as /etc/services can easily be maintained by *newfig*. It's channel definition only needs to provide the link between the channel and the file. If desired, a syntax checking step can be written and added to the channel definition:

```
%channel services {
    file /etc/services;
    syntax services-syntax;
    owner root;
    mode 444;
};
```

This channel can be used in the configuration to add lines to services, as follows:

```
rsyncd: {
    services "rsync 873/tcp";
    services "rsync 873/udp";
};
```

Since *newfig* generates files from scratch, the entire contents of the file must be specified by the configuration. This means that there must be a baseline of data available to add to the services channel to ensure that all the standard entries are there. Although each entry could be listed separately, it would be easier to include the baseline from a separate file (shown here with a broken line):

```
all: {
  %include services /opt/newfig/base/services;
};
```

cron

Each crontab file needs to be controlled as a separate channel. For most systems, only root and a few system crontabs need to be managed. Since it is unwise to edit a crontab file directly, this channel uses a proxy file instead. For the best results, the proxy should be persistent across invocations of *newfig* so that crontab itself is only invoked when there has actually been a change. For ease of demonstration, it is assumed that the macro PROXYFILES contains the name of a directory that holds proxy files.

This channel definition has to define different actions for each operating system it supports, as there is wide variation in the use of the crontab command. The channel also invokes a syntax checker to ensure the channel's contents are correct.

```
%channel cron.root {
    file "${PROXYFILES}/crontabs/root";
    syntax "syntax-cron";
    linux { action "crontab -u root -"; };
    sunos { action "crontab"; };
};
```

Typical usage of this channel would be:

```
sunos {
  cron.root "10 3 * * 0 /usr/lib/newsyslog";
};
```

sysctl

Linux *sysctl* is used to configure various kernel and driver parameters at runtime. The desired settings are kept in the file sysctl.conf and the command *sysctl* is run to set the parameters. *Newfig* can easily be used to drive the generation of sysctl.conf and provides central control over these settings. If a system is configured to be a web server, then its sysctl.conf can contain the settings needed to provide maximum performance.

If it is then changed to run a database server, *newfig* would alter the contents of sysctl.conf as described by its configuration to use the appropriate settings.

A sysctl channel for linux would probably look like this:

```
%channel sysctl {
  linux {
    file /etc/sysctl.conf;
    action /sbin/sysctl -p;
  };
};
```

The linux conditional is not strictly necessary, but does allow greater flexibility in the use of the channel. On non-linux systems, the channel contents will be ignored rather than generating an error.

Symbolic Links

Although *newfig* does not provide any built-in mechanisms for managing symbolic links, adding the functionality is as easy as writing a script. All that is needed is an action script that reads pathname pairs from standard input and ensures that one is a symbolic link to the other. This is a simple script to write, and it can be made as elaborate as necessary. It would also be beneficial for the script to have a syntax checking option.

An example channel definition for handling symbolic links is as follows:

```
%channel symlink {
    syntax "link-action -s";
    action "link-action";
};
```

This channel would be used as follows:

```
sunos {
  symlink /etc/inet/hosts /etc/hosts;
};
```

Management of an application's multiple versions can be handled via symbolic links as follows:

```
proftpd-1.2.9: {
  symlink /opt/proftpd-1.2.9 /opt/proftpd;
  resync /opt/proftpd-1.2.9;
};
proftpd-1.2.10rc3: {
  symlink /opt/proftpd-1.2.10rc3 /opt/proftpd;
  resync /opt/proftpd-1.2.10rc3;
};
```

File Distribution

Newfig does not have a built-in file distribution mechanism, not even for its own configuration files. The focus of this tool was on automatic configuration, so it relies on other means to accomplish file distribution. *Newfig* can easily be used to drive a file distribution and synchronization mechanism. For the sake of example, consider a script, named *resync*, that takes a list of directories and files on its standard input. Each entry is synced up with a central server using rsync [10] (recursively for directories). The following channel can be used to feed the data to *resync*:

```
%channel resync {
    action resync;
};

all: { resync /opt/local; };
samba: { resync /opt/samba; };
```

Inet Daemon

The typical inet daemon is controlled through the file /etc/inetd.conf. However, some systems (such as Linux) have a more sophisticated inetd that is configured through a collection of files in a directory, typically /etc/inetd.d . An infrastructure with a mix of these systems can still be controlled with *newfig* but the two types of inet daemons must be configured separately. This example uses a *directory channel* and a null channel. A directory channel contains a list of files and ensures that the target directory contains each of the named files and only those files. The files can originate anywhere but will bear the same names in the target directory. Here are two definitions, one for sunos (which uses the traditional inetd) and one for linux:

```
%channel inetd {
  sunos {
    file /etc/inetd.conf;
    owner root;
    mode 644;
    syntax "inetd-syntax";
    action "pkill -1 -u root inetd";
  };
};
%channel xinetd {
  linux {
    directory /etc/xinetd.d;
    action "pkill -1 -u root xinetd";
  };
};
```

Typical usage would look like this:

```
rsyncd: {
  inetd rsync stream tcp nowait
    root /usr/sbin/in.tcpd
    /usr/bin/rsync --daemon;
  xinetd /opt/rsync/etc/xinetd/rsync;
};
```

Note that the body of the rsyncd clause specifies data for both inetd and xinetd. However, it does not need to distinguish between different system types. That distinction is made in the channel definitions, not their

```
ftp-standalone: {
    rc /opt/ftp/rc/ftpd;
};
ftp-inetd: {
  inetd "ftp stream tcp nowait root /opt/ftp/sbin/ftpd ftpd -a";
};
```

Listing 1: ftp multiplexed between stand-alone and inetd service.

usage. Thus any system which supports xinetd can be added to the channel definition.

Differing FTP Usages

Because *newfig* evaluates logical clauses declaratively rather than procedurally, it is able to detect conflicts between clauses. For example, the following two clauses are conflicting and are flagged as an error:

```
one: two;
two: !one;
```

This feature can be used to protect a configuration from implementing conflicting configurations. One example of this ftp: it can be used either from within *inetd* or as a stand-alone daemon. Sometimes there are good reasons to use both types of configurations within the same infrastructure. *Newfig* can handle this and can also guard against accidentally attempting to implement both methods on the same system.

Assume that a configuration has channels to support the configuration of inetd.conf and boot time "rc" scripts in the style of System V. Listing 1 shows how ftp can be defined to act as stand-alone in some cases and as an inetd service in others.

Individual machines are configured to request either of these two symbols:

```
server1 -> ftp-standalone;
server2 -> ftp-inetd;
```

To prevent a misconfiguration from setting both symbols for the same server, the following statement can be used:

```
false: ftp-standalone & ftp-inetd;
```

Since the symbol *false* is always false, this statement will cause a logic conflict whenever both ftp-standalone and ftp-inetd are true.

Deployment at CNN

We began roll-out of *newfig* in to the CNN web farm infrastructure earlier this year. The web farm consists of over 800 hosts running a mix of Solaris and Linux. Prior to the use of *newfig*, we had constructed a system to control file distribution utilizing rsync. A central repository (gold server) holds all the files that need to be distributed for the various platforms we support. The *resync* script uses information about the host to determine a list of directories that must be kept in sync, then uses rsync to ensure that they are. This system, part of an effort called *Unity*, is used to distribute binaries and configuration files for key services in our infrastructure, including web service and ftp service as well as patches.

The deployment of *newfig* was built upon the success of resync. The initial configuration for *newfig* completely replaced the functionality of the original resync, and its deployment was seemless. Once *newfig* was in place, we targeted hosts.allow as our first file to control: it is relatively simple to support but requires significant fine-grained control.

A few weeks of effort was spent collecting the existing settings from across the infrastructure and codifying them in the *newfig* configuration language. Our initial goal was to automatically generate hosts. allow files that were no more restrictive than the ones already in place. In may cases the resulting files were more generous. Testing of this configuration was accomplished by generating the file as /etc/hosts.allow.x, then comparing the results to the existing hosts.allow. We eventually reached a point where the host access being removed could be summarized in a short list, and we decided that each of the items on the list was acceptable or even desirable. Then we changed the configuration to generate the actual hosts.allow file. Of the nearly 800 machines in the infrastructure, we only experienced access problems with one.

The resulting configuration was approximately 2400 lines spread out across 52 files, plus four separate files utilized in *include* statements. A filter script was written to organize and optimize the channel results as well as check for errors. We found many errors in the pre-existing hosts.allow files which had gone undetected, usually a class C network specification with no trailing dot. The next goal for hosts.allow is to analyze the current access and determine what is still needed and what should be removed. We expect to dramatically simplify the final configuration by rationalizing host access across large groups of systems.

The hosts.allow experiment was sufficient to prove the viability of the project. With its success we intend to take control over the hosts file, the startup scripts, crontab files, inetd.conf, boot time configurations (especially default routes), and perhaps passwd and shadow. The hosts file poses a unique challenge. Our hosts are divided in to internal and external, depending on whether or not they can be accessed directly by the outside world. All external hosts receive a common hosts file, but internal hosts use DNS. We have found it desirable to use minimal hosts files on the internal hosts: ones that only contain information on the host itself, and the NIS and NFS servers. We want *newfig* to generate this file from a list of hostnames, ensuring that the IP addresses are always correct and providing central control over the hosts that appear in the file.

Future Work

Our experiences with *newfig* are still very limited. As the support staff becomes more accustomed to its use, we plan to extend its control to as many facets of our systems as is feasible.

One set of files we would like to control with *newfig* are the passwd and shadow files. Currently we use NIS for portions of our infrastructure and nothing but local files for the remainder. Controlling access by system or by groups of systems is easy with NIS, but extremely difficult with static files. The tradeoffs with NIS are well known, including unsuitable security and

a poor level of robustness. We could replace NIS with a system based on LDAP, but in order to reduce single points of failure we do not want any of our externally facing servers dependent on a central service. *Newfig* would provide us with the centralized control that we need, but there are security issues that need to be considered for the distribution of the shadow information.

We want to control startup scripts with *newfig*, but these pose a number of interesting problems. Startup scripts differ widely among systems, requiring either separate channels or a single channel loaded with extra information. Effective control of the startup scripts would also require automatic stopping and starting of the daemons those scripts control as scripts are added and removed. Our goal is to control all startup scripts, so that those scripts which are not needed simply will not be included in the configuration. This goal is complicated by operating system vendor patches that alter these scripts.

We have found the use of "net" symbols (such as "net.10.1.2") to describe a system's local network very beneficial for many aspects of system configuration. For example, the symbols are the basis for determining if a system is "internal" or "external" (the latter are accessible by the outside world). However, the current implementation is very limited, and assumes that network divisions all fall on the traditional class C boundary [4]. The usefulness of these symbols could be enhanced by extending the notation to something more general, especially something that includes a netmask. But the simple boolean logic makes this more difficult. Further thought in this area would be beneficial.

Newfig provides a natural way to implement conformance for data in files. When a configuration change requires lines to be removed from files which are controlled by *newfig* the changes happen automatically as a result of file construction. However, this characteristic does not extend to the side effects implemented by action scripts, such as the symbolic link example in 6.5 and the file distribution example in 6.6. In these cases, *newfig* is constructing the input to a process which generates side effects. Consider the case of symbolic links with the following example:

```
one: alpha {
    symlink /opt/etc/one /etc/one;
};
```

The first time *newfig* runs, host *alpha* will have the symbolic link /etc/one created. If *alpha* is removed from the definition of the symbol *one* then the output line will no longer appear in the symlink channel. However, the symbolic link will remain as there is nothing that will remove it. We have an idea on how this problem can be overcome and will be pursuing its implementation.

There are times when it is advantageous to impose an ordering on output statements for a particular channel. Rather than use syntactic rules to imply an ordering, we envision a way to explicitly specify interrelationships between the output lines. One possibility is to allow for the specification of priorities on output statements, with a default of 100, and ensuring that lines appear in priority order. Thus a line that must always appear first could be given a priority of 1, and a line at the end a priority of 200.

The manipulation of macros in the configuration files is well motivated but poorly implemented. They are not as intuitive as one would hope. Further work needs to be done on this concept to provide the necessary functionality in a way that is still easy to follow.

Software Availability

The software was developed internally at CNN. Its availability for public use and review has not yet been determined.

Author Information

William LeFebvre is a technology fellow and David Snyder is a Chief Engineer. Both work for CNN Internet Technologies, the organization that runs over 60 web sites for CNN and Turner Broadcasting.

References

[1] Burgess, M., "Cfengine-Reference version 2.1.3," Centre of Science and Technology, Faculty of Engineering, Oslo College, Norway, Feb 2004.

[2] Craig, W. and P. McNeal, "Radmind: The Integration of Filesystem Integrity Checking with Filesystem Management" *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, p. 1, USENIX Association, 2003.

[3] Hagemark, B. and K. Zadeck, "Site: A Language and System for Configuring Many Computers as One Computer Site," *Proceedings of the Workshop on Large Installation Systems Administration III*, p. 1, USENIX Association CA, 1989.

[4] Harrenstien, K., M. Stahl, and E. Feinler, "DoD Internet Host Table Specification," *RFC 952*, October 1985.

[5] Osterlund, R., "PIKT: Problem Informant/Killer Tool," *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV)*, p. 147, USENIX Association, 2000.

[6] Roth, M., "Preventing Wheel Reinvention: The psgconf System Configuration Framework," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, USENIX Association, Berkeley, p. 205, 2003.

[7] Terpstra, J. and J. Vernooij, *The Official Samba-3 HOWTO and Reference Guide*, Prentice Hall, 2003.

[8] Thorton, J., "Prescriptions: A Language for Describing Software Configurations," Technical Report 94-18, Jun 1994.

[9] Traugott, S. and L. Brown, "Why Order Matters: Turing Equivalence in Automated Systems

Administration," *Proceedings of the Sixteenth Systems Administration Conference (LISA XVI)*, USENIX Association, p. 99, 2002.

[10] Tridgell, A. and P. Mackerras, "The Rsync Algorithm," Technical Report TR-CS-96-05, The Australian National University, June 1996.

[11] Venema, W., "TCP WRAPPER: Network monitoring, access control, and booby traps," *Proceedings of the Third UNIX Security Symposium*, p. 85, September 1992.

AIS: A Fast, Disk Space Efficient "Adaptable Installation System" Supporting Multitudes of Diverse Software Configurations

Sergei Mikhailov and Jonathan Stanton – George Washington University

ABSTRACT

Efficiently installing and configuring large sets of computer systems is an important concern for system and cluster administrators. Current solutions usually follow one of the two approaches: an image-based install or a metadata-based custom install. Both approaches limit the opportunities for optimizing the installation time by coupling the system specification with the installation technique and ignoring the relationships between configurations over time (as they evolve with patches and new packages).

The Adaptable Installation System (AIS) is a new model and implementation that attempts to address these shortcomings by taking a hybrid approach to client system installation. As in the metadata-based approach, it uses descriptors to express what the final system should look like in terms of composition and configuration. At the same time, it uses imaging for part of the client re-installation to achieve speed. In this paper we present the design and implementation of AIS along with details on the algorithm that builds images and performance results of running the prototype system on a set of RedHat based machines.

Introduction and Motivation

Efficiently installing and configuring large sets of computer systems is an important concern for system and cluster administrators. Numerous programs that facilitate automated and unattended installations have been created. Generally they follow one of the two approaches: an image-based install or a metadata-based custom install. Both techniques are widely used and effective in certain scenarios. However, they both have two main disadvantages:

1. They couple the specification of the system (golden-server, Kickstart or other configuration file) with the installation technique (over-network disk-image writing, package or OS install tool). This limits opportunities for algorithmic optimization and prevents using the best specification with the best technique in all cases.
2. They both assume a static model of system installation, meaning they assume installation is a singular, fairly heavy-weight event. Although some adaptation to more dynamic environments

Installation Factors

	Speed of Installation	Number of Configurations	Available Server Storage	Optimal Installation Approach
1	Not a factor	Small	Not a factor	Custom, Image
2	Not a factor	Small	Factor	Custom
3	Not a factor	Large	Not a factor	Custom
4	Not a factor	Large	Factor	Custom
5	Factor	Small	Not a factor	Image
6	Factor	Small	Factor	AIS
7	Factor	Large	Not a factor	AIS
8	Factor	Large	Factor	AIS

Table 1: Factors that affect installation scenarios.

such as Clusters-on-Demand [18] is possible, they do not capture the relationship between system configurations over time.

Table 1 compares the applicability of the two approaches with respect to three typical considerations in large installation/cluster management. The right-most column indicates the preferred installation approach given the values of the three considerations that are shown in the columns to the left. When the speed of client re-installation is not critical, custom install is a better option because of its ability to scale to a large number of system configurations and modest storage requirements. A new system configuration can be supported by creating a corresponding metadata descriptor, i.e., a Kickstart file. Imaging remains a preferred approach when the speed of client re-installation is critical. However, as the number of configurations increase dramatically, it becomes time consuming to manage and requires large amounts of storage on the server.

Both imaging and custom installation approaches fall short of simultaneously supporting fast client re-installations, large number of system configurations and moderate storage usage. The Adaptable Installation System (AIS) is a new model and implementation that attempts to address these shortcomings by taking a hybrid approach to client system installation. As in the metadata-based approach, it uses descriptors to express what the final system should look like in terms of composition and configuration. At the same time, it uses imaging for part of the client re-installation to achieve speed.

The need to support a large number of system configurations while simultaneously allowing fast client re-installations is motivated by the development of "utility" computing and "clustering on demand" services such as the Oceano [4, 10] and Cluster-On-Demand [18] projects. These projects' business model is that an organization manages computer clusters on behalf of its clients (whether internal or external) and guarantees an agreed upon level of performance. It maintains a large pool of client machines and dynamically reallocates them from one virtual cluster to another as necessary. A second motivation is the continued need to patch or upgrade, and occasionally

System Configuration
A description of the composition of computer system software in terms of the packages installed on this system. In the case of an RPM-based distribution of Linux, a list of all RPMs installed on this system.

System Configuration Descriptor (SCD)
An XML file that describes system configuration at a class level. Configuration is described in terms of Installation Base and additional packages that make up the system.

Installation Base
A minimal, working system configuration. Typically consists of a kernel, C libraries, and a small set of common Unix programs.

Host Configuration
System Configuration, plus any modifications to operating system configuration files needed to achieve the desired host state. Host configuration can be achieved manually, by copying host-specific files from the AIS server, or by running tools such as Cfengine [6].

Host Configuration Descriptor (HCD)
An XML file that specifies host configuration details. It references the SCD on which this host configuration is based.

AIS Server
The machine executing the AIS server code that calculates the contents of each image and builds the cached image files. This machine also hosts the image cache files and package files required for client installation.

Table 2: Definitions of system components.

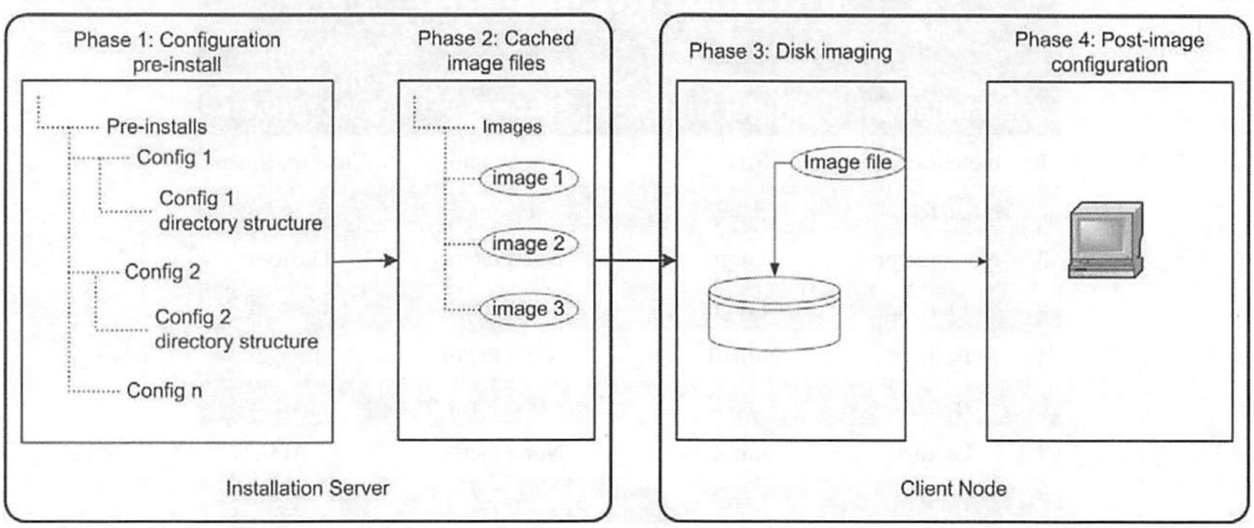


Figure 1: Stages in installation process.

downgrade, production servers. This cycle causes a continually increasing set of configurations that differ only by a few package versions. To maintain them as online images would require continually growing storage and management complexity.

```
<ImageConfiguration>
  <InstallBase>fc1</InstallBase>
  <OsBaseAlterations/>
  <Installations>
    <package type="rpm">mozilla-1.4.1-17</package>
    <package type="rpm">other package</package>
    ... ...
  </Installations>
</ImageConfiguration>
```

Figure 2: Sample system configuration descriptor.

```
<HostConfiguration>
  <ScdId>Configuration 1</ScdId>
  <ConfigFiles>
    <file perms="644">/etc/hosts</file>
    <file perms="644">/other/file</file>
    ... ...
  </ConfigFiles>
</HostConfiguration>
```

Figure 3: Sample host configuration file.

AIS Operation

To better understand how AIS operates, it is helpful to know the phases involved in client installation. They are shown in Figure 1. The terms that are used in the section are defined in Table 2.

As seen is Figure 1, AIS operates on two fronts: the installation server and the client machine being installed. On the AIS Server, AIS maintains a repository of System and Host Configuration Descriptors and a repository of installation packages. With this, AIS can reconstruct on disk any system configuration using its descriptors. This is shown as Phase 1 of the overall process in Figure 1. Note that AIS does not store or recover application data files except for host configuration contained in the HCD. The data files can be maintained on network file servers or separate data disks. One such method is described in [17]. Furthermore, AIS creates and maintains a repository of images (Phase 2), which it uses as the first step in reinstalling a client machine. This step is depicted as Phase 3 of the overall process.

The key to AIS operation is that it does not necessarily store images for all system configurations. Furthermore, an image does not have to correspond to any specific configuration. Instead, the content of the image cache is determined by an algorithm and the same image can be used for initiating a re-installation of more than one system configuration. The algorithm aims to minimize the client installation time. In the decision making process it considers all system configurations that might need to be installed and the disk space available for storing image files. Since the image does not necessarily correspond to an exact system configuration, AIS performs the necessary post imaging steps to arrive at the exact system configuration. This step is depicted as Phase 4 in Figure 1.

On the client, AIS is a script that runs as soon as the machine boots. The script uses a third party imaging tool, Frisbee, to retrieve the image from the AIS Server and write it to disk. After this, the script performs additional package installations, as necessary, to achieve the desired system configuration. Finally it performs any host specific configuration.

The following sections provide additional details about AIS operations on the Installation Server and the client machines.

AIS Operations on the AIS Server

Overview

The content of the image cache depends on all the System and Host Configuration Descriptors AIS is managing. The objective is to maintain such a combination of images that it minimizes the time it takes to install the next client machine, while meeting the disk space constraint. In the context of a large number of system configurations it is not feasible to store an image for every configuration. Thus, the images that are maintained do not necessarily correspond to a particular system configuration. Instead, an image can be

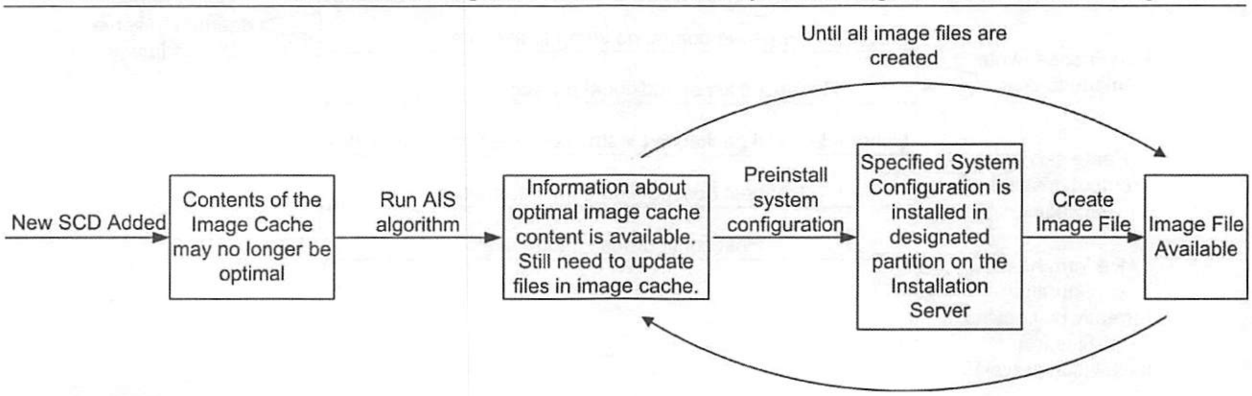


Figure 4: Adding a new system configuration descriptor requires image cache to be updated.

a mixture of common components from various configurations. This explains why in the post-image configuration phase AIS may need to install additional packages – to arrive at the desired system configuration from a generalized image configuration.

System and Host Configuration Descriptors

For AIS to manage a system configuration, a corresponding System Configuration Descriptor (SCD) must be created. A sample SCD is shown in Figure 2. A SCD specifies configuration of a system only in terms of the installed packages. Each SCD references an InstallBase, which is a minimal working system. The packages that make up the Installation Base plus the packages listed in the Installations section of an SCD constitute all the packages for a given system configuration. A system configuration specified by a SCD represents a class of systems. Any host specific configuration is contained in a Host Configuration Descriptor, HCD. A sample HCD is shown in Figure 3.

Introducing New System Configuration

Figure 4 shows the steps that are taken by AIS when a new System Configuration Descriptor is introduced. The cache content may no longer be optimal since when it was originally calculated the just-added SCD was not considered. This necessitates AIS to rerun the algorithm to determine the new content of the image cache. After the new content of the cache is determined, AIS needs to create the images. This is done by first installing the system configuration in a designated partition on the AIS Server and then running an image creation program. The last two steps are repeated for every image file.

The process of refreshing the image cache takes a considerable amount of time. After AIS determines how many configurations should be in the cache and what should be their content, each of those systems needs to be installed on a dedicated hard disk on the AIS Server in order to create an image. As such, refreshing of the image cache is meant to be an off-line operation scheduled during times when it can complete before starting to server client re-installation requests.

AIS Operations on the Client Machine

Image and Host Customization

On the client, AIS is a script that runs after the client node is booted. The steps involved in the client installation are shown in Figure 5.

The script contacts the AIS Server via HTTP with the client's MAC address. This allows AIS to uniquely identify the client and determine which system configuration should be installed and which image files should be used. The Frisbee server is started that serves the image file and AIS sends the client the command that it should execute to retrieve the image. After the image is written to disk, the client contacts the AIS Server to retrieve the list of any addition packages that might need to be installed to achieve the desired system configuration. After any additional packages are installed, the client script performs host specific configuration by copying OS configuration files from the AIS Server.

Algorithm Details

We will present a specific algorithm for constructing a set of *good* system images to cache. Many other algorithms also exist and we are continuing to investigate them. Each potential algorithm has a different set of tradeoffs. The one we present here is a fairly simple merge-based algorithm that maintains the invariant that all proposed images and targets are complete sets of packages with no missing dependencies. This algorithm produces results that are noticeably better then pure imaging or pure metadata based approaches, however, they are theoretically non-optimal (in the sense of minimizing the time to install any potential requested configuration).

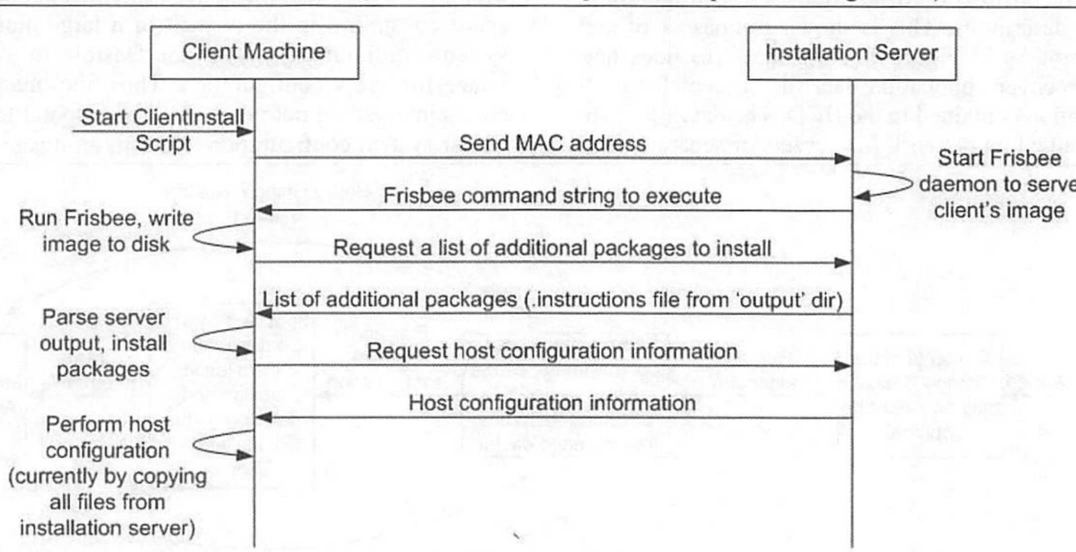


Figure 5: Client installation process.

The results of experiments conducted using the merge-based algorithm are further discussed later.

Partitioning Available Space Among Installation Bases

The algorithm operates on two levels. First, it determines how much disk space, out of all available for image caching, to allocate to each installation base. Since images are not compatible across installation bases, it is best that at least one image per installation base is available. The amount of space allocated to each installation base is proportional to the installation size of all systems configurations that rely on this base. For example, if the installation size of systems that rely on a given base is thirty percent of the total installation size of all systems AIS manages, then thirty percent of available disk space will be allocated for caching images of this base.

Determining the Composition of Cached Amalgam Images

Merge-based Algorithm

The second step of the algorithm determines the makeup and number of images to create within the space allocated for each installation base. Ideally, there would be enough space to store an image for every system configuration. However, in the context of a large number of configurations, this may not be feasible.

This step continuously merges the two selected configurations into one, until the remaining configurations fit in the allotted space. The two selected configurations are those that share a set of packages which also has the largest installation size. The resultant configuration after the merge is that common subset of packages.

Once this decision step completes, it produces three pieces of output:

1. A mapping of each system configuration to the image file that will be used in the imaging phase of client installation;
2. An SCD that describes the content of each of the images; and
3. A file that lists any additional packages that need to be installed for each system configuration.

If a particular system configuration was merged in, then there will be no image that represents it exactly. In this case the file will list the missing packages. For those systems that were not merged-in, the file will not list any packages.

Experimental Results

AIS is currently implemented in Python and works with RPM based Linux distributions. The existing Frisbee tools are used to generate compressed disk images and reliably multicast them to the client machines. The image times reported use Frisbee directly. Tables 4, 5, and 6 demonstrate the results of running AIS against ten system configurations. While this is not the large number of configurations AIS is intended to manage, the numbers provide strong evidence of the advantage AIS provides over other approaches. These tests were conducted on a set of X86 PC's with Pentium3 processors and a 100 Mbps Ethernet network. Although not the newest generation of PC's, we believe these are representative of actually deployed hardware.

The configurations used in this example are all based on the Fedora Core 1 Linux distribution. They vary in size and composition. The second column of Table 3 describes the content of each configuration in terms of how to achieve it using the redhat-config-packages program. redhat-config-packages, per default comps.xml file in RedHat/Fedora, divides all packages in five global groups: Desktops, Applications, Servers, Development and System. Each package group is further subdivided into subgroups. Each packages can be either mandatory, default or an optional member of a given subgroup. In Table 3 the package group name is

Configuration	Description	Image Used
Conf 1	[applications-]	{1}
Conf 2	[Applications+]	{9,8,6,4,2}
Conf 3	[applications-], [servers-]	{3}
Conf 4	[Applications+], [Servers+]	{9,8,6,4,2}
Conf 5	[applications-], [servers-], [development-]	{5}
Conf 6	[Applications+], [Server+], [Development+]	{9,8,6,4,2}
Conf 7	[applications-], [servers-], [development-], [kde]	{7}
Conf 8	[Applications+], [Servers+], [Development+], [System+]	{9,8,6,4,2}
Conf 9	[Applications+], [Servers+], [Development+], [System+], [Desktops+]	{9,8,6,4,2}
Conf 10	[Default]	{10}

Table 3: Configuration to image file mapping.

capitalized and followed by a plus sign only if every package of every subgroup is installed. The package group name is not capitalized and is followed by a minus sign if only mandatory and default packages of every subgroup are installed. Thus, Conf 5, for example, consists of only mandatory and default packages belonging to the subgroups in Applications, Servers, and Development package groups.

Table 3 shows that after this particular run of AIS, due to the disc constraints provided, the image cache consists of only six image files, as indicated by six unique entries in the rightmost column. For every system configuration in the leftmost column of Table 3 an image file that will be used during client re-installation is shown in the corresponding row of the rightmost column. For configurations 9, 8, 6, 4, and 2 there is no image in the cache that represents their exact configurations. The amalgam image, depicted as 9, 8, 6, 4, 2, will be used in the imaging phase of client installation for configurations 9, 8, 6, 4, and 2. For other system configurations, there is an image file that represents that configuration.

Table 4 shows the time it took to install each of the ten configurations using the three approaches. RedHat Kickstart was used for custom installation and Frisbee was used to derive the timing results for imaging approach. For those configurations that had a corresponding image in the cache under AIS, the time is nearly identical to Imaging. The one second delay is roughly how long it took to complete the host configuration, as can be seen in Table 6. Those configurations that used the amalgam image, took slightly longer than Imaging because of the additional packages that needed to be installed and host configuration. However, the Imaging approach required almost twice the disk space for storing image files as AIS, as seen in Table 5.

To further demonstrate interesting implications of AIS approach in maintaining a system configurations infrastructure, we compare the behavior of Imaging and AIS under evolving configurations in Figure 6. This can occur when the administrator wants to keep all versions of the same configuration as it evolves (patches applied, new packages installed) throughout its lifetime. In typical imaging approach, an image for every version of the configuration must be maintained, consuming much more storage than the size of the changes.

In Figure 6 AIS maintains only one image, with which it can achieve any version of configuration. As new versions are introduced, they may get incrementally longer to install, because they deviate more from the imaged configuration, as shown in Scenario 1. Alternatively, AIS can update the image so that installation of most recent versions takes the least time, as shown in Scenario 2.

Related Work

A number of tools allow for automated and unattended installations. Tools that allow metadata-based installations, among others, include RedHat Kickstart [15], FAI [7, 8], and LUI [13]. They can support large number of configurations, but take a relatively long time to install and couple the system specification and

	Conf 1	Conf 2	Conf 3	Conf 4	Conf 5	Conf 6	Conf 7	Conf 8	Conf 9	Conf 10
Kickstart	0:37:03	1:06:14	0:52:52	1:13:34	1:03:32	1:14:05	0:40:05	0:54:56	1:14:43	0:20:05
Imaging	0:03:22	0:05:13	0:04:46	0:05:19	0:04:25	0:05:20	0:04:44	0:05:18	0:05:23	0:02:36
AIS	0:03:23	0:05:14	0:04:47	0:07:49	0:04:26	0:08:44	0:04:45	0:09:25	0:09:51	0:02:37

Table 4: Timing of custom install, pure imaging and AIS approaches.

	Installation Pkg. Repository	Image Repository
Kickstart	1.8 GB	---
Imaging	0 or 1.8 GB	9.7 GB
AIS	1.8 GB	5.1 GB

Table 5: Storage requirements for three approaches.

	Conf 1	Conf 2	Conf 3	Conf 4	Conf 5	Conf 6	Conf 7	Conf 8	Conf 9	Conf 10
Imaging	0:03:22	0:05:13	0:04:46	0:05:14	0:04:25	0:05:21	0:04:44	0:05:21	0:05:22	0:02:36
Additional Installations	0:00:00	0:00:00	0:00:00	0:02:34	0:00:00	0:03:22	0:00:00	0:04:03	0:04:28	0:00:00
Host Configuration	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01
Total	0:03:23	0:05:14	0:04:47	0:07:49	0:04:26	0:08:44	0:04:45	0:09:25	0:09:51	0:02:37

Table 6: Detailed timing for AIS.

the installation technique. SystemImager [7, 9] and Frisbee [3] are tools that follow the imaging approach. As such, they install quickly, but lack flexibility in configuration and require disk space that increases with the number of system configuration variations.

A number of more sophisticated host configuration management tools exist which attempt to solve the larger problem of creating and maintaining software configurations across a large number of servers and desktops. These tools are discussed in [2, 1, 14, 5, 16, 12, 11] and are complementary to the AIS approach, as AIS only attempts to solve the problem of fast, efficient installation and could easily be matched with a SCM system for ongoing management. Currently the host configuration phase of the client installation process is achieved by copying over the necessary configuration files from the Installation Server. While this is sufficient for a prototype implementation, the overall value of the system would increase if one of the SCM systems was incorporated as one of the phases of the overall AIS process.

Availability

The AIS system as well as additional documentation is available for download under an open-source license at the web site http://www.ensl.cs.gwu.edu/projects/ais/ .

Future Work

AIS is a prototype implementation of a hybrid approach to installing software on a large number of client machines. The current implementation is limited to RPM. However, conceptually the ideas can carry over to other package management systems, as well as to using multiple package management systems on the same machine, if necessary. What is important is the

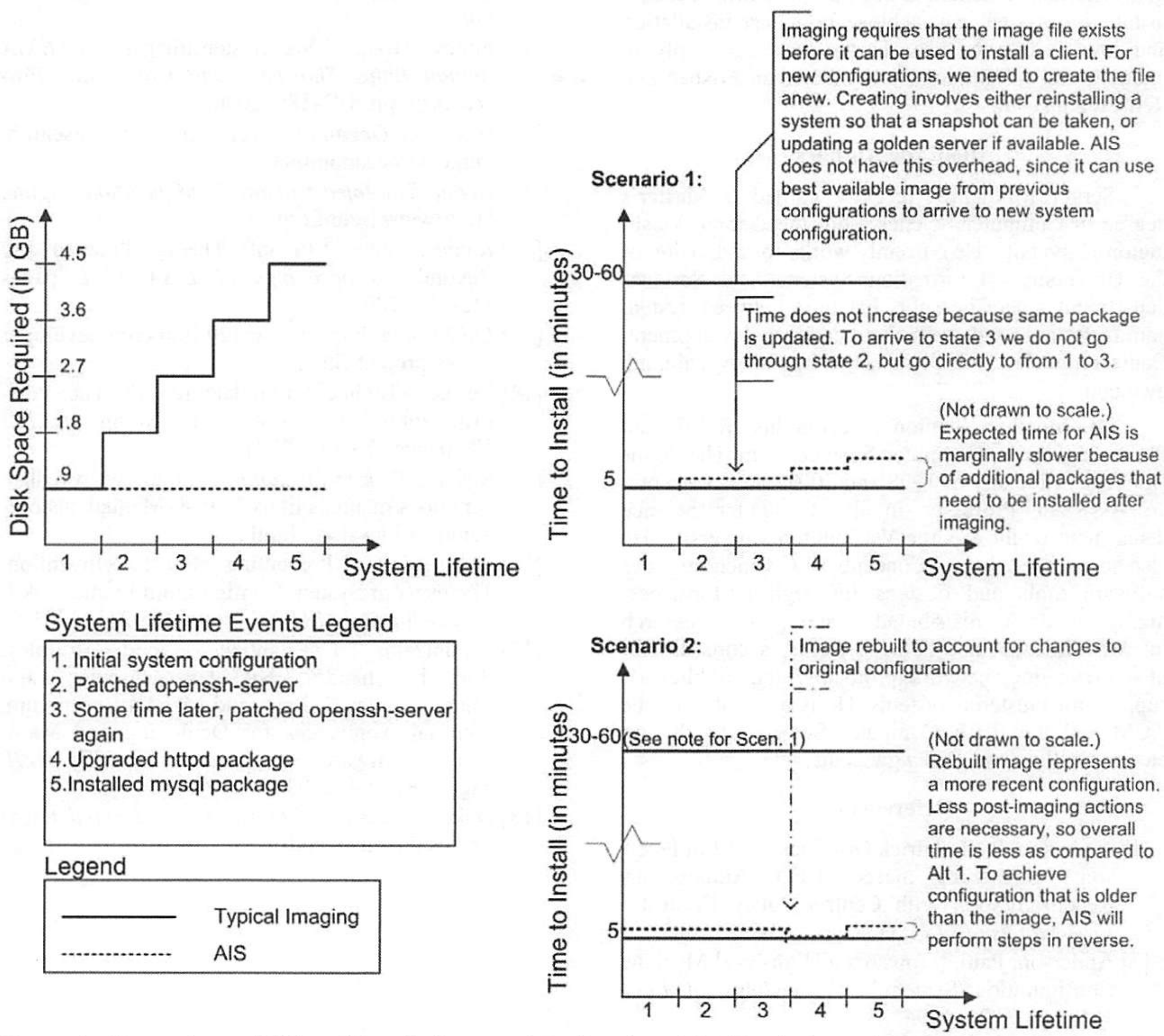


Figure 6: Comparison of disk and installation speed for Imaging and AIS solutions when the system configuration evolves.

ability to determine the makeup of the system's configuration in terms of installed packages, be it from one or more package management systems' databases.

Ability to query and list the content of the system is a significant advantage provided by package management systems; something that is not available by default in systems where a lot of software components are installed from source, "tar.gz" files or similar mechanisms. Furthermore, source installations do not provide dependency information and thus are not naturally suited for AIS-type systems, which construct system images from metadata descriptors.

Conclusion

AIS combines the features of custom install tools and imaging tools and provides a beneficial balance of ease of maintenance, scalability to large number of system configurations, and speed of client re-installation. This paper demonstrates how a hybrid, caching installation system can achieve both fast installation and low disk space use. These ideas can apply to imaging and installation tools other than Frisbee and RedHat Kickstart.

About the Authors

Sergei Mikhailov recently earned a Master's degree in Computer Science from the George Washington University. He currently works for a division of the University's Information Systems and Services department where he splits his time between system administration and web applications development. Contact him electronically at sergei.mikhailov@alumni.gwu.edu .

Dr. Jonathan Stanton received his M.S.E. and Ph.D. degrees in Computer Science from The Johns Hopkins University in 1998 and 2002. He is currently an Assistant Professor in the Computer Science department of the George Washington University. He also co-founded Spread Concepts LLC which provides software tools and designs for high performance, highly available distributed systems. His research interests include distributed systems, secure distributed messaging, network protocols, and middleware support for clustered systems. He is a member of the ACM and the IEEE Computer Society. Reach him electronically at jstanton@gwu.edu .

References

[1] Anderson, Paul, Patrick Goldsack, and Jim Paterson, "SmartFrog meets LCFG: Autonomous Reconfiguration with Central Policy Control," *Proceedings of LISA XVII*, pp. 213-222, 2003.

[2] Anderson, Paul, "Towards a High-level Machine Configuration System," *Proceedings of LISA VIII*, pp. 19-26, 1994.

[3] Hibler, Mark, Leigh Stoller, Jay Lepreau, Robert Ricci, Chad Barb, "Fast, Scalable Disk Imaging with Frisbee," *USENIX Annual Technical Conference Proceedings*, pp. 283-296, 2003.

[4] Appleby, K., "Oceano – SLA Based Management of a Computing Utility," *Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[5] Anderson, Paul and Alistair Scobie, "Large Scale Linux Configuration with LCFG," *Proceedings of the Atlanta Linux Showcase*, pages 363-372, http://www.lcfg.org/doc/ALS2000.pdf, 2000.

[6] Burgess, Mark, "Cfengine: A Site Configuration Engine," *USENIX Computing Systems*, Vol. 8, Num. 3, http://www.cfengine.org/ , 1995.

[7] *Enterprise Infrastructure Workshop Notes*, LISA 03 Workshop, http://www.infrastructures.org/workshop/ , 2003.

[8] *FAI: Fully Automatic Installation for Debian GNU/Linux*, http://www.informatik.uni-koeln.de/fai/ .

[9] Finley, Brian, "VA SystemImager," *USENIX Annual Linux Showcase and Conference Proceedings*, pp. 181-186, 2000.

[10] IBM, *The Oceano Project*, http://www.research.ibm.com/oceanoproject/ .

[11] *Isconf: The Infrastructure Configuration Engine*, http://www.isconf.org/ .

[12] Kanies, Luke, "ISconf: Theory, Practice and Beyond," *Proceedings of LISA XVII*, pages 115-123, 2003.

[13] *Lui Project*, http://www-124.ibm.com/developerworks/projects/lui/ .

[14] Oetiker, Tobias, "TemplateTree II: The Post-Installation Setup Tool," *Proceedings of LISA XV*, pages 170-186, 2001.

[15] RedHat, *Kickstart installations*, http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/ch-kickstart2.html .

[16] Roth, Mark, "Preventing Wheel Reinvention: The psgconf System Configuration Framework," *Proceedings of LISA XV*, pages 205-211, 2003.

[17] Sapuntzakis, Constantine, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum, "Virtual Appliances for Deploying and Maintaining Software," *Proceedings of LISA XVII*, pages 181-194, 2003.

[18] Duke University, *COD: Cluster-on-demand*, http://issg.cs.duke.edu/cod/ .

autoMAC: A Tool for Automating Network Moves, Adds, and Changes

Christopher J. Tengi – Princeton University
James M. Roberts – Tufts University
Joseph R. Crouthamel, Chris M. Miller, and Christopher M. Sanchez – Princeton University

ABSTRACT

It is often difficult and time-consuming to manage computer 'moves, adds, and changes' that take place in a switched, subnetted environment. It is even more difficult when the accepted network policy requires that a computer be configured and always connected to the network on a specific Virtual LAN (VLAN) based on its usage. An on-going problem is to keep all of the network host information up-to-date, and to ensure that hosts always land on the correct subnet when they are plugged into switch ports. Utilizing some freely-available tools, as well as some home-grown software, we have built a system that automates a number of the tasks associated with moves, adds, and changes.

Where We Were

The Department of Computer Science at Princeton University has a routed IP network of 1500+ hosts, with a VLAN assigned to each subnet. IP addresses are assigned either from Princeton's address space or from any of a number of private IP address blocks, as defined in RFC 1918.[1] Hosts with private IP addresses are subject to Network Address Translation (NAT) by a CheckPoint firewall before any of their network traffic is sent out to the rest of the campus or the Internet. All intra-departmental routing is handled by a Foundry FastIron 1500 ethernet switch.

The department has a role-based network model, meaning that a host's place on the network is determined by its function and who uses it. Each host has a statically-assigned IP address tied to its ethernet MAC address. The network model specifies how hosts should be mapped to IP subnets, and how subnets correspond to VLANs. A faculty member's office machine belongs on the faculty-office VLAN. A host used by a member of the administrative staff belongs on the admin-staff VLAN. A graduate student's office machine belongs on the grad-office VLAN, while a host used in that graduate student's research belongs on a VLAN specific to the project. Figure 1 shows VLAN assignments for some of the roles a host might fit into.

[1]Address Allocation for Private Internets

Note that the main purpose of our network model is to segregate hosts by usage. In general, there are no restrictions on traffic between IP subnets. However, as some of the subnets are in private IP address space, access to hosts on those subnets is restricted from outside of the CS department.

As mentioned above, routing within the CS department is handled by a Foundry FastIron 1500 ethernet switch, which is at the core of our network. A number of other switches, from various vendors, are attached to it. They are configured as either "infrastructure" switches or "user" switches. The infrastructure switches are used for connections to servers and other network devices, such as printers, IEEE 802.11b [7] access points and network cameras. Infrastructure switch ports are in physically secure areas, preventing users from connecting to them. The user switches are used for network connections to end-user hosts. Most of the time spent dealing with ports is devoted to user switch ports.

There are a number of routine tasks involved in managing this network, including: host registration, maintaining compliance with the network model, and management of physical network connectivity. Prior to the implementation of autoMAC, these tasks required a lot of time and intervention by the Computer Science technical staff.

Class	Membership	Subnet	VLAN
Printers	printers	192.168.xx.0/24	19xx
Devices	cameras, env. monitors	192.168.yy.0/24	19yy
Faculty	faculty office hosts	128.112.aa.0/24	aa
Grads	graduate student office hosts	128.112.bb.0/23	bb
Graphics	graphics lab hosts	192.168.zz.0/24	19zz
PlanetLab	PlanetLab nodes	128.112.ccc.64/26	ccc1

Figure 1: Example network model host roles.

Host registration was accomplished using our inventory/host database system that was built around a Berkeley DB file, processed by a number of perl scripts. Hosts were entered into the system by a script that called 'vi,' allowing edits to be made on a template host entry. The same script allowed changes to be made to existing entries. Since the data entry environment used a simple text editor, it was difficult to assure accuracy and consistency of the entries. Once the required changes had been made to the DB file, DNS, DHCP, and NIS configuration files were generated and installed by using the 'make' command. Note that all of the preceding steps were carried out by a member of the technical staff.

In the process of entering or changing a host entry, it was necessary to examine the supplied information and make a decision about which VLAN the host should be assigned to. One problem with this method is that, given the same information, different technical staff members sometimes made different decisions, resulting in hosts with the same roles being assigned to different VLANs. Consequently, some hosts ended up where they didn't belong.

Port configuration changes on the switches required an administrator to connect to the appropriate switch and to change its settings manually. If needed, a patch cable would be installed to connect the switch port to an office wall box. When a user moved a machine from one location to another, it was very likely that either a change in the switch configuration or change on the patch panel (or both) would be required.

The process of registering a host and getting it connected to the network, or moving it to a different VLAN because its role in the network model changed, frequently involved multiple email exchanges between the requesting user and the technical staff. Often, the initial request would be something like "Please register my new computer on the network with the name mongo." As this was not *nearly* enough information to process the request, we would send a reply asking for the rest of the information we needed. It might take several rounds of email replies to get all of the information needed to register the host and to activate a wall box jack.

Including the time spent awaiting additional information from the user, it could be hours from when the initial request was received before a new host could be used on the network. The only semi-automated task was the generation and installation of the configuration files.

Another time consuming task was isolating a host that was the source of a network problem. It was necessary to locate which switch port it was using and to reconfigure the port manually. Some local command-line tools were available to assist in identifying the problem host, but it was still necessary to connect to the appropriate switch and reconfigure the port in order to move the host to a different subnet for isolation or testing.

While packages are available to manage host registration (such as NetReg [11, 10]) and switch configuration (such as Splat [2]), no freely available package ties everything together and automates all of the tasks our staff is required to do in order to manage how hosts connect to the network.

What We Did

The system we envisioned and then implemented allows users to register new hosts using a web interface, with minimal intervention by the technical staff. The request is processed automatically by a daemon, and the new host is typically available for use on the network within 10-15 minutes. The host can be connected to any "public" ethernet jack and will automatically be placed on the proper VLAN. Public ethernet jacks include those in open spaces as well as departmental offices and labs. In general, any ethernet jack that is accessible to people outside of the technical staff is considered public.

Access to the web form is restricted to members of the CS department by requiring that they enter their departmental username and password. The form presents the user with questions to determine the proper class and VLAN of the new host. Infrastructure VLANs used by secured service hosts are not available on the switch ports the users can access. In addition, we are in the process of implementing an enhancement that will offer to each user in the department only their allowed user classes.

The idea of a web interface for host registration is not new. What makes autoMAC different is that it enables automated control of how a host is connected to the network *after* registration.

Automatic VLAN Assignment

A key feature of the autoMAC system is that a user can plug their host into any public ethernet jack attached to one of our user switches, and it will be automatically connected to the correct VLAN. This is accomplished using a RADIUS [6] server that "authenticates" the MAC address of the host attempting to connect to the network.

A number of web-based and IEEE 802.1X [9] authentication/authorization systems exist that require information to be manually entered or supplicant software to be installed on a host before it can access the network. These systems are well suited for authorizing user access to a network, but are not designed to work with unattended devices, such as printers or network cameras. In addition, requiring our users to install IEEE 802.1X supplicant software before they access the network is not practical, given a stream of frequent, temporary visitors. Ideally, for our purposes, it should be possible to connect any type of ethernet

device to any switch port and have it placed on the correct VLAN without any additional human interaction. Using a MAC address to identify a host, rather than a name and password to identify a user, makes this possible. Our users are still required to authenticate with their username and password to access departmental servers.

The RADIUS server configuration is built from data in our host database. For each MAC address, a "user" entry is generated, using the MAC address as both the username and password. The entries contain additional tags that specify to which VLAN a host belongs.

A host whose MAC address is not listed in the host database is considered unknown, and is therefore not listed in the RADIUS configuration file. When a user switch makes an authentication request with an unknown address, the RADIUS server responds in the negative. The switch then sets the port to which the host is connected to a registration VLAN, where the NetReg server presents our web-based host registration interface to the user when a web browser is started.

If a user moves a host from one wall box jack to another anywhere in the Computer Science building, the host stays on the correct VLAN since the switch port behind it is automatically configured using information provided by the RADIUS server. To move a host to a different VLAN, we simply need to change its IP address in the host database to an address on the other VLAN and rebuild the configuration files. The next time the host is connected to any switch port in the network, it is automatically placed on the new VLAN. If we need to force the issue, we can use tools we have developed to identify the switch port currently being used by the host, and turn the port off and back on. This forces the switch to re-authenticate the host the next time the host sends out any network traffic, causing it to be moved to the new VLAN.

Implementation Details

This system was not built from scratch. Rather, it was constructed by modifying our existing tools, and combining them with a number of open source packages available on the Internet. Specifically, we used FreeRADIUS [5] and NetReg, in addition to our existing host database and new code written to work with our web server and the above packages.

As mentioned earlier, our host database system is built around a Berkeley DB file. The vi-based data entry interface allows far too much latitude in what may be entered for a given field, making field validation difficult. However, the system is implemented with perl scripts that use a common library of functions to manipulate the data in the DB file. Having this library makes it fairly easy to implement new functionality such as bulk or daemon-based data entry.

The development of autoMAC began in July 2003, when we decided to implement a FreeRADIUS server to control access to our IEEE 802.11b wireless infrastructure. The Cisco access points (APs) we were using could be configured to query a RADIUS server using a MAC address for the username and password. The AP would then either allow or deny association of the client machine based on the response of the RADIUS server. While we realize that this is not strong security, it did, along with turning off broadcast of the SSID, prevent casual association with our wireless infrastructure.

While we were working on the FreeRADIUS server, we checked to see what other pieces of our network infrastructure might be able to make use of a RADIUS server. We discovered that the Foundry switches could utilize a RADIUS server to authenticate users, and configure their ports to specific VLANs. If this functionality could be extended to use MAC addresses for usernames and passwords, it would give us a *very* useful tool.

We spoke to a number of ethernet switch vendors to express our interest in MAC-based "authentication" for their switches, and received positive responses from several of them. As our user switches are from Foundry, we encouraged them to add this feature to their FastIron line, which they did. We have subsequently learned that several other switch vendors also now offer this feature, or will be offering it soon.

One of our goals for this project was to simplify the task of host registration for both our user base and the systems staff. To this end, we investigated two web-based host registration systems: Southwestern University's NetReg [11] and Carnegie Mellon University's NetReg/NetMon [10]. Both of these systems had features that we liked, but they also did much more than we needed. In the end, we built our own NetReg, utilizing some of the code from Southwestern's system which they graciously released under the GNU GPL [3].

Our NetReg implementation consists of a Linux system running the Internet Systems Consortium's BIND (DNS) and DHCP software, configured according to the instructions provided with Southwestern's NetReg system. The Linux system is also running the Apache Web Server, configured to use SSL and serve our custom registration page. The machine is connected to a dedicated registration VLAN as well as one of our production infrastructure VLANs. The infrastructure connection allows communication with our existing host database system. The machine uses an 'ipchains' firewall to limit vulnerability to attack from the registration VLAN and has routing disabled to prevent traffic from leaving the VLAN.

Tying Things Together

Once all of the parts of the puzzle had been identified, we needed to piece them together into a usable, cohesive system. We had a host database that associated

MAC addresses with IP addresses. We wanted a web-based system to add entries to the database. We had switches that could set ports to VLANs based on data from a RADIUS server, and we had a RADIUS server that could send the data. Now we needed to write some code to tie the components together.

Entering data in a text editor, based on email received from users, was an error-prone process. It was also a job we were looking to get out of. A web-based interface with field validation seemed to be a very good idea for a replacement.

The front-end we implemented allows users and support staff to easily enter all of the information required to register a new host and submit the request. Users of this interface are required to authenticate themselves with the same username and password used for connecting to our Solaris and Linux systems, as well as our e-mail server. We use SSL encryption to prevent user credentials from traveling over the network in the clear. All of the web pages are generated using PHP [4] scripts.

Figure 2 shows the host registration form. Extensive field validation is done to prevent duplicate entries and to ensure compliance with our network model. When the user submits the form, the request is sent to a new-host daemon that does the actual host registration and configuration file generation. The request is sent as a specially-formatted email message on behalf of the user filling in the form. The daemon parses the message body and invokes a perl script to update the host database DB file. It then runs a 'make' to generate and install all of our configuration files.

RFC 2868[2] defines RADIUS attributes to be used for "compulsory tunneling in dial-up networks." These attributes are leveraged by IEEE 802.1X (in Annex D) and RFC 3580,[3] which specify additional tunnel attribute information that can be used to assign VLANs to hosts being authenticated by a RADIUS server. In a VLAN environment, these attributes are returned as

[2]RADIUS Attributes for Tunnel Protocol Support

[3]IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines

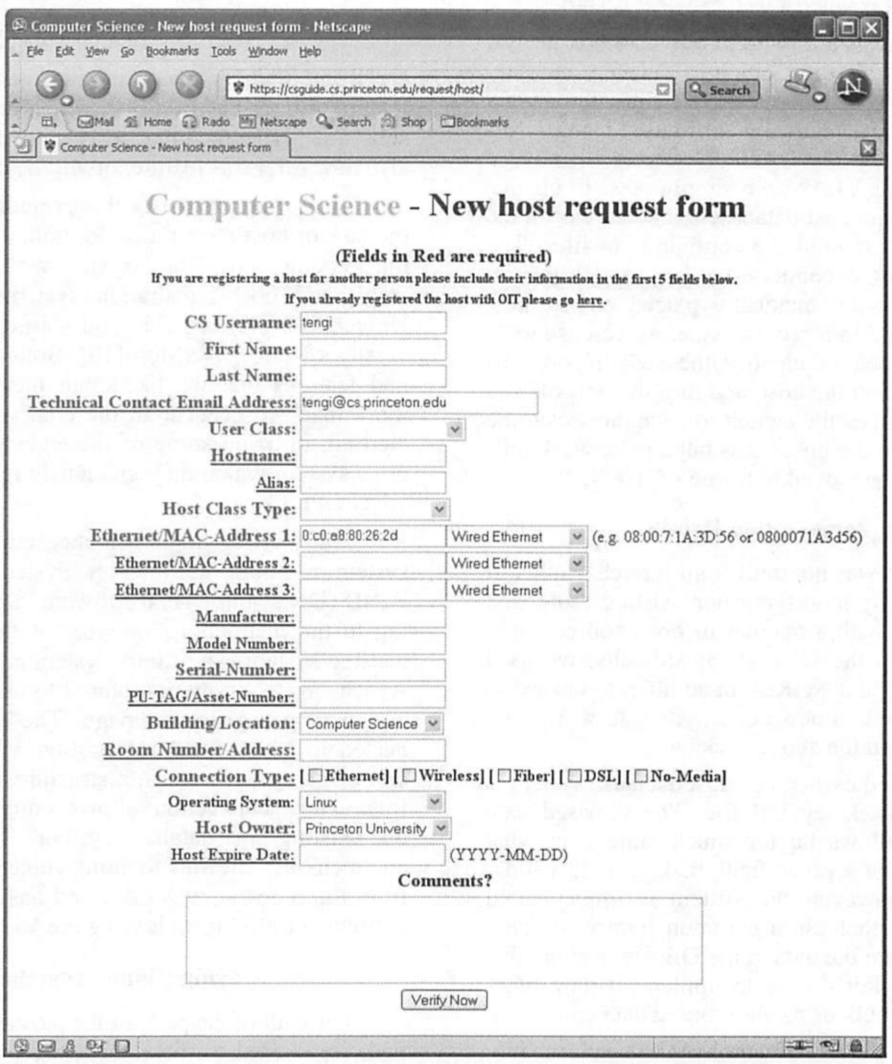


Figure 2: Host request form with default information filled in.

part of the 'Access-Accept' message from the RADIUS server when access is granted to a host, and can be used by the switch to configure a port's VLAN.

We generate a 'users' file for the RADIUS server that contains entries for every host registered in our database. There is a VLAN-based look-aside table used in this process to determine whether or not tunnel attributes should be added to the entry. Figure 3 shows an example RADIUS user entry with tunnel attributes while Figure 4 shows one without.

One of the VLANs that is not in the look-aside table is the one we use for 802.11b wireless access. Therefore, a host registered for wireless access would be listed as in Figure 4, without tunnel attributes. This is the type of entry we started with when we first implemented the RADIUS server for use with our 802.11b Access Points.

When a host is connected to one of our user switch ports, the switch sends an Access-Request message to the RADIUS server with the host's MAC address as the username and password. The RADIUS server returns either an Access-Accept or an Access-Reject message to the switch, based on whether or not the MAC address is known. If an Access-Reject is returned, the switch sets the port to the registration VLAN so that the user may register the machine using NetReg. The switch will also set the port to the registration VLAN if an Access-Accept is received without any VLAN information included. On the other hand, if the RADIUS server recognizes the address and has VLAN information for that address, it returns an Access-Accept message with the VLAN information which the switch then uses to set the port to the correct VLAN.

When a host is connected to the registration VLAN, its DHCP request is answered by the NetReg server. The reply specifies the NetReg server itself as the router for the host to use. Therefore, any subsequent IP traffic from the host will be sent to the NetReg server. The only other devices with IP addresses on this VLAN are other hosts awaiting registration, so it is unlikely that any network traffic will leak out from this VLAN to the rest of the campus or the Internet.

The DHCP reply also specifies the NetReg server as the DNS server to use. The DNS daemon code is configured to return the NetReg server's IP address for any lookup received. This means that hosts on the registration VLAN will be directed to the NetReg server for any web sites they may try to contact.

The Apache HTTP daemon is configured to present a login page for any unknown URL. This login page tells the user that they are using an unregistered host, and prompts them for their Computer Science username and password so that they may register the machine. This limits requests to members of the CS department. If someone is visiting the department for a few days and wants to use their laptop on the network, it needs to be registered by the member of the department they are visiting.

Upon successful authentication, the user is redirected to the host registration form shown in Figure 2. Submitting the form sends the registration request to the new-host daemon and displays a page to the user, stating that the registration request is being processed and instructing the user to disconnect from the network port and reconnect in ten minutes. When the host is disconnected or rebooted, the RADIUS authentication process starts all over again. This effectively removes a host from the registration VLAN once it has been successfully registered.

The email address to which the registration information is mailed uses procmail [12] to validate the message and save it in a spool directory. The new-host daemon, mentioned earlier, processes messages from the spool directory and adds the new hosts to the host database. The daemon then runs the 'make' command to generate and install all of the required configuration files.

An Example Scenario

Here then is an example scenario, showing how all of the parts operate together. The assumptions are that a Computer Science user wants to register a new host on the network; that the host will try to get an IP address with DHCP; and that the host will try more than once to obtain an IP address. Switch ports are configured to be on an unused, isolated VLAN until the switch receives information from the RADIUS server. Note that the level of detail in the following list items will vary, to prevent us from getting bogged down in low level details.

- The user connects the new host to a public ethernet jack, which is connected to a port on one of our "user" ethernet switches, and turns on power to the machine. The machine attempts to obtain an IP address using DHCP.
- The ethernet switch blocks the DHCP request, extracts the host's MAC address from the request

```
00d0b7b600ee    Auth-Type := Local, User-Password == "00d0b7b600ee"
          Tunnel-Type = VLAN,
          Tunnel-Medium-type = 802,
          Tunnel-Private-group-ID = 702,
          Fall-Through = Yes
```

Figure 3: RADIUS user entry with tunnel attributes.

```
00c04f7f1006    Auth-Type := Local, User-Password == "00c04f7f1006"
          Fall-Through = Yes
```

Figure 4: RADIUS user entry without tunnel attributes.

and uses the MAC address in an Access-Request message to the RADIUS server.

- Since this is a new host, the RADIUS server's configuration files do not list the MAC address, so it returns an Access-Reject message to the switch, which then configures the port to be on the registration VLAN.
- Because the switch blocked the host's first DHCP request, the DHCP client process on the host times out and sends another request.
- The NetReg server on the registration VLAN receives the DHCP request from the host and responds with a short-lived IP address on the registration subnet, as well as default router and DNS server addresses pointing to the NetReg server. The host uses this information to configure its network interface.
- The user starts a web browser on the host and attempts to connect to some web site. This causes the host to send a DNS query for the host named in the URL. We assume that the URL will be for 'HTTP' or 'HTTPS,' and that the host portion will contain a domain name, and not an IP address. The DNS daemon on the NetReg server resolves the domain name portion of the URL to its own IP address, and sends the address back to the host. If an IP address is used, the HTTP request will time out, as the request will not be forwarded off of the registration VLAN.
- The web browser on the host now makes a connection to the HTTP daemon on the NetReg server, requesting whatever page was specified in the URL. The HTTP daemon responds with an information page notifying the user that he is using an unregistered host. The page includes a link to the host registration web page.
- The user clicks on the link, and is prompted for their CS username and password by the HTTP daemon. If they enter valid credentials, the HTTP daemon sends the host registration form to the host. Otherwise, an "access denied" message is sent, and the user needs to try again.
- The user fills out the form, and clicks the "Verify Now" button. A CGI script on the NetReg server validates the entry data and, if all is well, sends a display-only page to the user with a "Submit Now" button at the bottom. Otherwise, it displays a partially completed registration form, with instructions on how to correct any errors.
- The user clicks on the "Submit Now" button, sending the form information to another CGI script which formats and sends an e-mail message, on behalf of the user, to the new-host daemon. The script also displays a page to the user, stating that the registration request has been sent and that the host should be rebooted in ten minutes.
- The new-host daemon parses the e-mail message, validates the information, and if all is well, selects an IP address and adds the host to the host database. The daemon then runs 'make' to generate and install the configuration files.
- After ten minutes, the user reboots his host. When the host reboots, it drops ethernet link, causing the switch to remove the port from the registration VLAN. The host then attempts to obtain an IP address using DHCP.
- The ethernet switch blocks the DHCP request, extracts the host's MAC address from the request and uses the MAC address in an Access-Request message to the RADIUS server.
- The RADIUS server now has the MAC addresses listed in its configuration files, along with VLAN information, so it sends an Access-Accept message, including the VLAN information, to the switch, which then configures the port to that VLAN.
- Because the switch blocked the host's first DHCP request, the DHCP client process on the host times out and sends another request.
- This time, the DHCP request is received by our production DHCP server, which responds with a reply containing the host's registered IP address, along with our normal DNS server and default router information.
- The user may now use his host normally on the CS network. All future reboots and DHCP requests from this host will result in the switch port to which it is connected being configured to the proper VLAN, as the MAC address is now known.

Security Considerations

While autoMAC limits user host connections to the appropriate VLAN (according to our role-based network model), we do not rely on the system to provide strong security. Intentional spoofing of MAC addresses is not addressed, as this possibility always exists.

Anti-spoofing ACLs on our core switch prevent hosts from pretending to originate from different subnets. As we mentioned earlier, users still have to authenticate with login name and password to access our departmental servers. Multiple attempts to register different MAC addresses from the same port within a short period of time could be logged and trigger a report.

If a host gets a new ethernet address due to a hardware change, the user will need to register the new ethernet address before he can use anything other than the registration VLAN. Ethernet card swaps between machines are most likely to occur within a user class (e.g., graduate student to graduate student, not graduate student to faculty), so the worst problem is one of host identification, not VLAN access. In the case of a graduate student swapping the ethernet cards of a research host and an office host, the machines will

end up on the wrong VLANs, and the student will no longer be able to use either machine effectively.

In the event that a host needs to be quarantined due to virus infection or other problems, it can easily be moved to an isolation VLAN. The only way for a user to circumvent that isolation would be to change the host's MAC address. If the new MAC address was unregistered, the host would be connected to the registration VLAN. If the MAC address of another registered host was used, the infected machine would be connected to the VLAN associated with the stolen MAC address. If the MAC address was already in use, the result would be two machines getting the same IP address from the DHCP server, and neither machine would work reliably. This would most likely trigger a complaint to the technical staff from the owner of the host whose MAC address had been stolen.

Future Enhancements

With our new system it is relatively easy to reconfigure hosts to be on different VLANs. As a result, we think it should not be too difficult to integrate this with other systems that scan automatically for active viruses and current patch levels before a registered host is allowed network access. In addition, infected or vulnerable hosts on the network can be quarantined, and transferred to isolated VLANs where the appropriate host patching tools are available.

According to the documentation, our Cisco 802.11b APs can also make use of the RADIUS tunnel tags to specify a VLAN for a wireless client. Adding tunnel tag support to the APs will allow us to implement a registration VLAN for wireless users as well as wired users. It will also allow us to segregate wireless traffic by host role, as we do for wired traffic. Currently, all wireless users are on the same VLAN.

Availability

Some of the tools we have developed are available to the public, so that others may implement similar systems. The host database system we use will not be made available, as we are in the process of replacing it. The software and some configuration examples can be found at http://www.CS.Princeton.EDU/autoMAC/ .

RFC documents can be found on the Internet Engineering Task Force web site http://www.ietf.org/ .

Conclusions

In a network environment such as ours, where hosts from different VLANs can be plugged into any public ethernet jack, it can be difficult and time-consuming to ensure that the switch ports behind those wall jacks are always on the correct VLAN for a given host. Using a number of different tools and some switch firmware features, we have put together a system that uses the host MAC address to automatically configure switch ports properly.

Combining automatic port configuration with web-based automated host registration allows users to start using new hosts on the network with little or no intervention by the technical staff. This means that if a user brings a new host into the department after normal business hours, he won't have to wait until the next day to start using it. When paper deadlines are looming, and our students need to add a few machines to the network in order to complete their experiments (at 3 a.m.), near-instant host registration is a big win.

By implementing this system, we have moved from a mostly manual to a mostly automated method of handling network moves, adds, and changes. This has improved response time to our users and also provided support staff with time to work on other projects. Additionally, we have improved compliance with our network model, by automating the switch-port-to-VLAN mapping. This project would not have been possible without the availability of a number of Open Source Software projects.

Author Biographies

Chris Tengi is a System and Network Administrator in the Princeton University Computer Science Department. His professional interests include network architecture and management, and just about any new networking technology. He can be reached at tengi@CS.Princeton.EDU .

James M. Roberts is the Director of Computing for the School of Engineering at Tufts University. He can be reached at jmr@cs.tufts.edu .

Joseph R. Crouthamel, Chris M. Miller, and Christopher M. Sanchez are System and Network Administrators in the Princeton University Computer Science Department. They can be reached, respectively at jrc@CS.Princeton.EDU , cmmiller@CS.Princeton.EDU , and cmsanche@CS.Princeton.EDU .

References

[1] Aboba, B., "IANA Considerations for RADIUS (Remote Authentication Dial In User Service)," *Network Working Group Request for Comments 3575*, Internet Engineering Task Force, *Update of [6]*, July 2003.

[2] Abrahamson, Cary, Michael Blodgett, Adam Kunen, Nathan Mueller, and David Parter, "Splat: A Network Switch/Port Configuration Management Tool," *Proceedings of the 17th Large Installation Systems Administration Conference (LISA-03)*, pp. 247-256, USENIX Association, October 2003.

[3] Free Software Foundation, *GNU General Public License*, http://www.gnu.org/licenses/licenses.html .

[4] The PHP Group, *PHP scripting language*, http://www.php.net/ .

[5] The FreeRADIUS Project, *FreeRADIUS Server Project*, http://www.freeradius.org/ .

[6] Rigney, C., S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," *Network Working Group Request for Comments 2865*, Internet Engineering Task Force, *Updated by [13] and [1]*, June 2000.

[7] IEEE Computer Society, "Higher-Speed Physical Layer Extension in the 2.4 GHz Band," *Std 802.11B-1999*, IEEE, *Supplement to [8]*, 1999.

[8] IEEE Computer Society, "Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, *Std 802.11*, IEEE, 1999.

[9] IEEE Computer Society, "Port-Based Network Access Control," *Std 802.1X-2001*, IEEE, 2001.

[10] Carnegie Mellon University, *NetReg/NetMon*, http://www.net.cmu.edu/netreg/ .

[11] Valian, Peter and Todd Watson, "NetReg: An Automated DHCP Registration System," *Proceedings of the 13th Conference on Systems Administration (LISA-99)*, pp. 139-148, USENIX Association, November 1999.

[12] van den Berg, Stephen R., Philip Guenther, et al., *procmail Mail Processing Suite*, http://www.procmail.org/ .

[13] Zorn, G., D. Leifer, A. Rubens, J. Shriver, M. Holdrege, and I. Goyret, "RADIUS Attributes for Tunnel Protocol Support," *Network Working Group Request for Comments 2868*, Internet Engineering Task Force, *Update of [6]*, June 2000.

More Netflow Tools: For Performance and Security

Carrie Gates, Michael Collins, Michael Duggan, Andrew Kompanek and Mark Thomas –
Carnegie Mellon University

ABSTRACT

Analysis of network traffic is becoming increasingly important, not just for determining network characteristics and anticipating requirements, but also for security analysis. Several tool sets have been developed to perform analysis of flow-level network traffic, however none have had security as the primary goal of the analysis, nor has performance been a key consideration.

In this paper we present a suite of tools for network traffic collection and analysis based on Cisco NetFlow. The two primary design considerations were performance and the ability to build richer models of traffic for security analysis. Thus the data structures and code have been optimized for use on very large networks with a large number of flows. Data filter rates are approximately 80 million records in less than 1.5 minutes on a Sun 4800.

Introduction

Cisco NetFlow [11] is becoming an increasingly popular method for analyzing network traffic, and several tools (e.g., [3, 8, 1]) have been developed to take advantage of this flow information. However, most of these tools have been developed within the context of academic settings, where performance was not critical. The SiLK Suite[1] was developed to provide analysis tools for very large installations, such as large corporations, government organizations, and backbone service providers. These sites often transfer large volumes of data, much of it extraneous (e.g., worm traffic, scanning activity).

In addition to having performance as a key element of the SiLK Suite, the tool set was developed with security analysis as a primary goal. This has facilitated the development of a new suite of tools that allow information filtering in a manner unavailable in other tool sets. This suite of tools has been field tested at a large ISP, and is now in operational use at this site. For example, the tool set is able to process approximately 80 million records in less than a minute and a half on a Sun 4800.

This paper provides an introduction to the SiLK Suite tool set, describing both the collection system and the analysis tools. It provides examples of how to use the analysis tools and the types of analyses that can be performed. SiLK is then compared to related tools.

The SiLK Suite can be down-loaded from http://silktools.sourceforge.net/ [5].

Overview of the SiLK Suite

The SiLK Suite consists of two primary components: the collection system and the analysis tool set. The collection system converts Cisco NetFlow Version 5 PDUs into a compressed binary format. The tools work on these compressed records, allowing a user to filter data in a variety of ways and to use a series of command line tools for data summarization.

SiLK was originally designed to address a problem inherent in traffic analysis: traditional payload-based analysis can make accurate judgments with a relatively small amount of data, but traffic analysis requires larger volumes of data to assess trends and large scale behaviors. Coupled with the volume of data seen on our client network, the amount of traffic summaries received was on the order of tens of Gigabytes a day.

In order to manage this volume of traffic, SiLK adopted three strategies: the footprint of individual records was reduced to the minimum necessary to store security-relevant information and nothing more, files were split into several common pre-defined categories to reduce the amount of time to look for specific traffic, and the SiLK Suite was made gzip-transparent. SiLK reads gzipped or unzipped files transparently, which yields a substantial performance bonus (in our experience, gunzipping a file in memory is cheaper than loading the unzipped file).

Collection System

The collection system has been designed to minimize the amount of disk space required to store data, while still supplying the data required for security analysis of network traffic. The collection system takes Cisco NetFlow Version 5 PDUs and converts them to a "packed" format. The packed records are stored in a hierarchy based on the router class (e.g., ingoing, outgoing) where this information is specified by the type of record (e.g., in, inweb, out and outweb), and date and time, with hourly files available at the leaves. A separate file is maintained for the flow

[1] SiLK stands for System for Internet Level Knowledge, with the SLK capitalized in memory of Suresh L. Konda, who was the founder of the project.

records from each router. Only flows that have been routed are recorded – flows representing traffic that was dropped by the router due to an access control list (ACL) violation are not saved.[2] A NetFlow record consists of 48 bytes. We reduce this record size to achieve disk storage savings via three approaches:

- do not store the fields that are not required
- reduce the number of bits used to store some information
- remove the storage of fields where the file hierarchy can indicate the same information

The first approach results in the removal of eight fields from the NetFlow record. In particular, information about the network path is not maintained. This results in the removal of the following fields: input interface number, output interface number, the source AS number, the destination AS number, the source mask, the destination mask, and the next hop IP number. In addition, the type of service information is not kept. This results in a savings of 19 bytes per record.[3] Additional space savings comes from a reduction in the number of bits used to store various information.

For example, time information is only stored with accuracy to within one second, rather than the millisecond precision provided by NetFlow. In addition, the packed record only uses 12 bits to store the start time of the flow, and 11 bits to store the elapsed time of the flow. The header for the packed data file contains the start time for all of the records in the file. As each file contains only one hour of flow data, the start time only needs to be the number of seconds since the start of the file. The end time is actually the number of seconds elapsed since the start of the flow. By default, NetFlow flushes a continuous flow after 30 minutes, and so the elapsed time requires one fewer bit than the start time (based on this default). If a site is using a different configuration (in particular, flushing less frequently than 30 minutes), then the code will need to be modified to accommodate this difference.

Other bit savings come from storing the average bytes per packet, rather than the absolute number of bytes. There are 14 bits dedicated to the number of bytes per packet, and an additional six bits to represent the fractional portion of the value. Additionally, only 20 bits are used to represent the number of packets in a flow. If this value overflows, then an overflow bit is set. Therefore we only have accuracy in this field up to approximately one million packets. After this, we use a multiplier to achieve greater values, but at the cost of accuracy. It is important to note that implicit in this design is the concept that flows with small payloads are more important or interesting than larger ones, and hence we are not concerned with the exact values for larger flows. However, the overflow bit has been designed to still provide information (by acting as a multiplier), rather than being used as an error flag.

As noted above, some information has been removed from the data record and is maintained by the structure of the underlying file system. For example, the directory hierarchy specifies the type of record as the second level in the hierarchy, where the type of record can be either web or non-web. For many large networks, the majority of traffic consists of web-based traffic. By splitting out web traffic into a separate location, the number of bits required to store port information for web records can be reduced. That is, the ephemeral port information is maintained (16 bits), while only two bits are used to represent the web port in use (where the web port can only be one of 80, 443 or 8080). A third bit is used to indicate if the web port is the source port or destination port. In addition, since all web traffic uses the TCP protocol, there is no need to store the protocol information. In total, these changes result in a savings of 21 bits, which is a savings of 2-bytes per record in disk storage. For a site that sees ten million web flows per hour (incoming or outgoing), there is a savings of nearly 500 MB per day.

Using all these techniques results in a flow stored as source IP address, destination IP address, source port, destination port, protocol, flag combination, start time, elapsed time, number of packets, and the bytes per flow (which is converted to number of bytes by the analysis tools). The packed record only requires 22 bytes of on-disk storage, while the original NetFlow PDU requires 48 bytes. For the web data, this is reduced even further to 20 bytes per record. For sites that experience large volumes of traffic, this can result in significant savings in disk space.

Analysis Tools

Data Manipulation

The SiLK Suite currently provides 13 tools and seven associated utilities. Libraries are provided for reading the packed records, which performs file globbing (using fglob library calls, which is how file globbing will be referred to for short) based on information provided through a standard set of arguments. These arguments specify the start and end date/time, the type of data (e.g., incoming or outgoing, web or non-web), and the sensor (router). These flags can be provided to the tools that read the packed records, and are used to specify exactly what files are read. This allows for enhanced performance by reducing the amount of traffic that needs to be searched, as only the relevant portions of data are examined. (In other tools, all traffic is maintained as flat files, where the analytical tools require the user to specify the input data file. The directory hierarchy employed by SiLK, however, is incorporated into the analysis tools, allowing the analyst to focus on the behaviour of interest, rather than

[2]We currently do process flows that encounter ACL violations on one of our client sites, where these records are saved in a different directory in the file hierarchy, however this code is not yet available in the open source release.

[3]Future versions might incorporate some of these values, however the open source version currently does not save this information.

needing to find the appropriate file. Additionally, the hierarchy allows the tools to more quickly locate small files containing the information for the time and sensor of interest.) The tools default to using incoming traffic, both non-web and web, for all sensors. Additionally, if only a day is provided for the start date, then the default is to process the entire day. If an hour is provided, then the default is to process only that hour. To process some other amount of time (e.g., two hours), the end date flag is required. If no start date is provided, the tools default to using the data available so far for the current day.

The primary tool is rwfilter.[4] This tool reads the packed data and can filter based on various options. These options include filtering based on the start or end time of the flow, the duration of the flow, the source or destination ports, the protocol, the number of bytes or packets, or the flag combination (for TCP). Perhaps one of the most useful options that can be provided for filtering is the source and destination addresses, which can be provided as single addresses, ranges of addresses, or as a set of unrelated IPs (called an ipset and described more fully below). Alternatively, all IPs that are NOT in the set provided can also be used. In addition to the base command line functionality, rwfilter has the ability to incorporate a user-compiled dynamic library. This library can be used to filter records based upon criteria that are too complicated to express on the command line, to perform "canned" queries more quickly than the command line would allow, and to perform stateful operations on large sets of flow records. For example, a user can define their own set of important services (e.g., dns/tcp, dns/udp, web, other tcp, etc.), using the dynamic library to count the number of flows for each of these services, and printing the results at the end of the rwfilter command.

The rwfilter tool provides two output options: --pass and --fail. The --pass option allows all data meeting the specified filtering options to be saved to a file (or, alternatively, stdout can be specified here, if the results are to be piped through another command), while the --fail option saves all those records that did NOT meet the filtering criteria. Both options can be used at the same time, allowing a user to chain rwfilter commands, where data that meets a condition can be saved in a file via --pass, while those records that fail the condition can be piped into another rwfilter command via --fail=stdout. The data files that are generated by rwfilter are in a binary format similar to that generated by the packing system, and which can also be read by the rw commands, which allows commands to easily be chained together. (Records output from rwfilter no longer have the contextual information provided by the file hierarchy and therefore fully expand fields such as start time. The result is a homogeneous stream of 32-byte records.)

A major functionality provided by this tool set is a binary representation of IP addresses, called an ipset. The ipset data structure is effectively a dynamically expanding checklist: the core of the structure is a list pointing to 65,536 8 KB bitmaps, where each bit indicates the presence of an IP address. Under normal circumstances, only a small number of the bitmaps are allocated, and most ipsets end up being less than a megabyte in size. However, the structure is very fast (any address is looked up in two memory loads) and consequently allows a user to query arbitrary sets of IP addresses as fast as any other query in SiLK.

An ipset can be built from an ASCII list of dotted-quad IP addresses using the buildset command. It is also possible to use the results from an rwfilter command to generate an ipset by piping the output from rwfilter to rwset. The rwset command reads in data in the packed format and generates an ipset of either the source IP addresses or destination IP addresses, as specified by a command line option.

The ipset files that result from buildset or rwset can be read using the readset command, which can print the IP addresses in the set, a count of the IP addresses, or various statistical information. Ipsets can also be combined using standard set functions such as intersect (setintersect) or union (rwset-union).

Ipsets allow a user to filter data on IP addresses that need not have anything in common (e.g., does not need to be in a range). For example, a site can maintain a "bad list" of IP addresses (addresses that are known to be malicious) as an ipset. This set can then be used to filter incoming traffic for any activity from these IPs, or to filter outgoing traffic to see if there is any communication to these IPs. If a second bad list needed to be added to the first, then the two could be merged using rwset-union. Similarly, to see what addresses the two lists had in common, setintersect could be used.

In order to view the records returned by rwfilter or similar utilities (e.g., rwsort, which is described below), the command rwcut must be used. This command reads in any packed data file, and prints the fields in the packed record, along with the sensor ID and the end time of the flow. The fields to be printed can be specified with the --field option, where the fields are numbered from one to 12. (Numbers were used to save the user from needing to type each required field in full, e.g., --field=sip,dip,sport,dport,stime. Additionally, once the user has memorized which numbers map to which fields, it allows them to easily specify ranges, e.g., --field=1-4,9.) The number of records to print can also be specified with the --num-recs option.

All of the tools work on packed data, as this is the most efficient. Given the large number of records that need to be handled quickly, Unix-like utilities,

[4] We use rw as a short-hand for raw. This is a historical convention, and refers to the type of packed files we are using and the type of data we are receiving.

such as rwuniq and rwsort were developed that use packed data. The rwuniq tool will return the unique entries for the specified field, along with a count, and so is equivalent to the Unix command uniq -c. In addition, rwuniq allows the user to set a threshold, so that only those entries that occur more often than the threshold are returned. The rwsort tool sorts packed data on the specified fields (allowing the user to specify a primary key and secondary key), outputting the results in a packed data format. Operations using these tools on packed data perform more than twice as fast as the same operations on plain text using traditional UNIX tools.

Currently, rwsort is limited to 50 million records. If the input contains more than 50 million records, sort proceeds based on just the first 50 million records. This limit was provided based on memory restrictions, and can be changed easily by modifying the source code. This design decision was made based on the assumption of 2 GB of RAM being available, and with the desire to provide the user with a consistent memory limit, rather than one that might change based on machine or machine usage.

Data Summarization

Other tools are intended to assist in traffic analysis by providing summarizations appropriate for graphing, and various statistical reports. For example, rwcount provides the number of bytes, packets and flows seen in the packed data provided, broken into user-specified time intervals (e.g., five minutes, one hour). This allows a user to glance at a report and determine if there was any sudden spike in activity. The tool rwtotal allows even finer granularity based on user-specified criteria. For example, a user can perform an rwfilter to extract all traffic going to a particular /24 address space, then pipe this to rwtotal and group the results (number of bytes, packets and flows) by the last octet of the destination address. This provides a count of all the traffic going to each IP address in a /24 network. Other than specifying various parts of the source or destination address (e.g., last 8 bits, last 16 bits), the user can also print results based on protocol, source port, destination port, number of packets or number of bytes.

The tool rwaddrcount is similar to rwtotal, however it is based on IP addresses instead of time intervals. It can take as input the results from an rwfilter query, and will return the total number of bytes, packets and flows for each source IP address, along with the time stamps for the first and last flows observed. The information provided can be further refined through command line options specifying the minimum and maximum flows, packets or bytes observed.

The tool rwstats computes a variety of statistics, based on the traffic flows provided to it. The number of flows, and the percent of the input that these flows represent, are provided for the groups specified by the user. Top N lists (e.g., sort the results by the number of flow records, and then only return the first N, such as 10, from the list – thereby presenting the results with the most traffic) can be provided by either source or destination IP address, where the user specifies the value for N. In this manner, the user can view the top 10 (for example) sources that generated the most number of flows to the monitored network, or the top 20 destinations that generated the most flows to outside addresses. This feature can also be used based on ports, rather than IP addresses. Additionally, the bottom N (those groups with the least number of flows) can also be specified. If preferred, the user can look at combinations of items, such as the top source-destination IP address combinations or source-destination port pairs. Additional statistics, such as the minimum, maximum, quartile, and interval statistics for bytes, packets and bytes/packet can be determined based on protocol (e.g., TCP, UDP).

In addition to the 13 tools provided by the SiLK Suite, there are also seven utilities. The utilities differ from the tools as they were provided to assist in some analysis tasks, based on user feedback. In contrast, the tools were designed to perform the actual analysis. The utilities are:

1. num2dot: This utility converts IP addresses from a 32-bit integer to dotted quad notation. It expects output from rwcut (where the rwcut IP format had been specified to be 32-bit integers, rather than the default of dotted-quad), with the fields to be converted specified on the command line. This tool is useful if output that contains IP information in both 32-bit integer and dotted-quad notation is desired. One example of where this would be useful is if the resulting flows needed to be imported into a spreadsheet. Using rwcut with --field=1,9,1-8 --integer-ips generates rwcut output with IP numbers as 32-bit values. num2dot can then convert fields three and four to dotted-quad notation, leaving the first field (source IP address) as a 32-bit integer, allowing easy sorting in the spreadsheet, yet still providing the dotted-quad value for the user (the third field would now contain the dotted-quad version of the source IP, while the fourth field contained the destination IP in dotted-quad).
2. rwappend: This utility appends new flow records to an existing packed file.
3. rwcat: This utility will concatenate packed files into a single stream.
4. rwfileinfo: This utility reads the header information of a packed file and prints it to the screen. This information includes items such as the number of records in the file and the command line that generated the file.
5. rwfglob: This utility can be used to determine what files will be processed given a set of fglob

options (e.g., start date, incoming or outgoing, etc.).

6. mapsid: This utility determines the sensor name or number, and can convert between the two representations.
7. rwswapbytes: This utility can be used to change the endianness of a packed file.

Security Analysis

These tools can easily be scripted to deliver regular reports. For example, one of the client sites using these tools produces top 10 lists on a nightly bases. The top 10 IP addresses that have seen the most flows, or bytes, or packets can easily be determined through a combination of rwfilter and rwaddrcount. Similar statistics can easily be generated for ports based on rwtotal.

The following sections demonstrate some of the capabilities of the tool set to detect various types of activity. All IP addresses have been obfuscated. All internal addresses are represented as 10.x.x.x, while all external addresses are represented as 241.x.x.x. Access to the data may be made available by special request to the authors.

Scanning Activity

Adversaries often perform a scan of a network as the prelude to an attack. In particular, "script kiddies" (unskilled attackers) will often deploy an exploit across all of the machines in a network [7]. This type of activity will appear as a SYN scan (in the case of a TCP-based exploit), where there might be some further communication with internal systems that respond to the scanner with a SYN-ACK. The SiLK tool set can be used to find scanners of this type, and to determine if particular machines should be investigated for compromise.

For example, to look for "fast" scanners (that is, scanners who have contacted a large number of machines in a short amount of time), we can do the following:

```
rwfilter --start=2004/6/29:17 \
         --syn=1 --ack=0 --fin=0 \
         --proto=6 --pass=stdout | \
rwaddrcount --print-recs \
         --rec-min=65000
```

The rwfilter command here uses incoming traffic (both web and non-web) by default. It processes one hour of data (17:00-18:00 GMT on June 29, 2004), looking for all flows where the SYN flag was set, and the ACK and FIN flags were not set. The other flags (RST, URG and PSH) can take any value. Only the TCP protocol is used (--proto=6, using the standard protocol numbers as defined by the Internet Assigned Numbers Authority (IANA) [4]). The results from the rwfilter command are passed through stdout to the rwaddrcount command. This command prints all the source IPs that had more than 65000 flows, along with the number of bytes, packets and flows, with start and end times (see Figure 1). There were two sources that met these criteria.

The source IP that had the most records (and who therefore presumably scanned the most targets) was 241.27.240.226, with 74,773 flows (a little over one /16 network, if each flow is to a different destination IP), while 241.21.21.24 had 65,732 flows. We therefore elect to examine the traffic from both source IPs in detail. Some of the information that we would like to know include how many unique destination IP addresses did each source target, and what ports they targeted.

To answer the first question regarding the number of destinations, we can extract all of the flows for each source via an rwfilter call. In this case we will save the results to disk so that we do not need to continually process an entire hour of data. We also drop the restriction on the flag combinations so that we see all of the TCP flows from this source. The command used for the first source IP address is:

```
rwfilter --start=2004/6/29:17 \
         --saddr=241.27.240.226 \
         --proto=6 \
         --pass=rwdatafile
```

The process is the same for the second IP address. The resulting file contains 75,199 records (obtained by using rwfileinfo, or by adding the option --print-stat to the rwfilter command). To determine the number of unique destination IP addresses in this file, we can use the command:

```
rwuniq --field=2 --no-title \
       rwdatafile | wc
```

which performs the equivalent of sort | uniq -c using the destination IP field (field = 2), with the titles for the fields turned off. The result was 75,199 lines, which indicates that there were that many unique IP addresses – indicating only one flow per destination IP address. However, given that there were more records observed when the flag restriction was dropped, there was likely some further communication between some of the additional destination IP addresses.

To determine the ports that were targeted, we can perform the same query as above, but replace the field value with 4 (for destination port). The result from this query (rwuniq --field=4 rwdatafile) is:

```
dPort  count
  80   75199
```

This shows that all of the flows were to destination port 80 (web). Similarly, running the same

IP Address	Bytes	Packets	Records	Start Time	End Time
241.21.21.24	3855940	87635	65732	06/29/2004 17:00:00	06/29/2004 17:48:13
241.27.240.226	5267496	109792	74773	06/29/2004 17:00:02	06/29/2004 17:49:40

Figure 1: The results from rwaddrcount.

commands for source IP address 241.21.21.24 also showed a scan of port 80.

We are interested in determining if there were any responses to these two scans. To determine this, we first create an ipset from these two IPs. We can do this by creating a file that contains the two IP addresses and then running buildset. Alternatively, we can do the following:

```
rwfilter --start=2004/6/29:17 \
      --syn=1 --ack=0 --fin=0 \
      --proto=6 --pass=stdout | \
rwaddrcount --print-ip \
      --rec-min=65000 --no-title | \
buildset stdin ip.set
```

With only two IP addresses, it is quicker to just create a temporary file, however if there had been a large number of IP addresses, than the latter approach is preferable.

We can then use the ipset that we have created as a filter on the outgoing data to determine what communication there was from the internal network to these two scanning IP addresses. The command we would use is:

```
rwfilter --type=out,outweb \
      --start=2004/6/29:17 --proto=6 \
      --dipset=ip.set --pass=rwdata.out
```

The result was a file consisting of 17,542 records. This indicates a very large number of responses! However, we are only interested in positive responses, or records where there was no RST returned to the source. The rwdata.out file can be further filtered on the flag combinations, to examine only those flows that contained no RST using the command:

```
rwfilter --rst=0 rwdata.out \
      --pass=rwdata.out.noRST
```

Unfortunately, this still resulted in 16,865 records. We therefore take a quick look at the data to see if we can determine anything interesting. We do this by first sorting on the source IP address (the internal responding host), followed by displaying the results:

```
rwsort --field=1 rwdata.out.noRST \
  | rwcut --field=1,2,3,4,6,7,8 \
  | less
```

A sample from the result set is given in Figure 2. This shows that the scanner was proceeding in order through the IP space. In this instance, the source had actually stumbled onto a honey-pot, which is why there was a response from each IP address in that particular subnet. In general, if an unusually high number of the same service is seen on the same subnet (e.g., 16000 web servers on a /16) where the subnet is a general network (that is, not a server farm, for example), then it might indicate a honey pot or a firewall (as some firewalls can be configured to respond in this manner). This hypothesis is further supported by each server responding with exactly seven packets and 1646 bytes, implying that they are returning the same content (or at least content that is exactly the same size!). In our case, it turns out that the majority of responses to the scan were due to this honeypot.

Worm Attacks

Recently, two prominent worms (Korgo [9] and Sasser [10]) have been released that scan port 445. When a vulnerable machine is found, each of the worms exploits the vulnerability, but then diverge to perform different activities on the infected machine. As we care less about external machines scanning our network for vulnerabilities than we do about internal machines that have been infected, we turn our attention to examining outgoing network traffic. We know that infected machines scan for vulnerabilities on port 445, so we can narrow our search by examining only flows with destination port 445. Since we are looking for machines that perform scanning of this port, by definition there will be a large number of unique destination IP addresses contacted by a single source. We therefore want to find all internal machines that are contacting large numbers of external machines on port 445. (Note that just a large number of flows is not necessarily indicative of an infection, but that a large number of unique destination IP addresses is more indicative.)

To extract the information we want, we first perform an rwfilter, and then pipe these results through a call to rwstats:

```
rwfilter --type=out \
         --start=2004/6/29:17 \
         --proto=6 --dport=445 \
         --pass=stdout | \
rwstats --pair-topn=10
```

As we are now examining outgoing traffic, we need to specify that the type is out instead of the default of incoming. We do not need to examine

sIP	dIP	sPort	dPort	packets	bytes	flags
10.10.10.1	241.37.150.226	80	1542	7	1646	FS PA
10.10.10.2	241.37.150.226	80	1543	7	1646	FS PA
10.10.10.3	241.37.150.226	80	1544	7	1646	FS PA
10.10.10.4	241.37.150.226	80	1545	7	1646	FS PA
10.10.10.5	241.37.150.226	80	1546	7	1646	FS PA
10.10.10.6	241.37.150.226	80	1547	7	1646	FS PA
10.10.10.7	241.37.150.226	80	1548	7	1646	FS PA
10.10.10.8	241.37.150.226	80	1549	7	1646	FS PA
10.10.10.9	241.37.150.226	80	1550	7	1646	FS PA
10.10.10.10	241.37.150.226	80	1551	6	1152	FS PA

Figure 2: Output from filtering on a particular destination IP.

outweb, as port 445 is not one of the web ports. Again we look at only one hour of data, extracting all traffic to destination port 445 using the TCP protocol. The output from this command is piped into rwstats, which produces the top ten source-destination IP pairs based on the number of records. The output from this command is give in Figure 3.

This is not exactly the output that we want, since we want the sources that have contacted the most destinations, not the source-destination pairs that have the most flow records. To get this information, we can specify a threshold on the number of flows that a source-destination pair must have before printing it to the screen. By specifying a threshold of one, we extract all pairs. However, this still only provides a list of all pairs, along with information about each pair such as the number of flow records. We can take this information and pipe it through some standard unix utilities to extract, for example, the ten sources who contacted the most destinations. The command to do this is:

```
rwfilter --type=out \
    --start=2004/6/29:17 \
    --proto=6 --dport=445 \
    --pass=stdout | \
rwstats --pair-top-threshold=1 | \
gawk -F"|" '{print $1}' | sort | \
uniq -c | sort -nr | head
```

The results from this command are:

```
78443 10.101.100.10
16083 10.123.100.100
  940 10.150.100.100
  127 10.115.100.100
   92 10.10.10.100
   43 10.20.1.20
   12 10.111.20.30
    9 10.30.100.40
    6 10.177.30.50
    5 10.199.100.60
```

This shows that four IP addresses contacted more than 100 unique destinations in a single hour, which is an unusually high number of destinations. (It has been observed by Williamson that workstations usually contact no more than ten destination IP addresses per hour [13].) These four machines therefore warrant additional investigation as they might be infected with Sasser or Korgo (or some other worm or virus). The only IP address that shows up in both top ten lists – that of number of connections to unique destination IP addresses and that of number of flows between it and some other source – is 10.150.100.100.

SYN Flooding

Another security concern is denial of service attacks. One of the common network-based denial of service attacks is SYN flooding. We can use commands similar to those used to detect worms to detect if a SYN flood has occurred. In this case, we want to detect all source-destination IP pairs that have seen an excessive number of SYN packets. To do this, we first filter on all incoming traffic for flows with the SYN bit set, but with no ACK or FIN, examining only the TCP protocol. We then run rwstats on the result, looking for the source-destination pairs that have the most flows. In fact, we can specify that at least some X number of flows are required before we consider this a SYN flood that we want to investigate. In this case, we choose $X = 1000$, resulting in the following command:

```
rwfilter --syn=1 --ack=0 \
         --fin=0 \
         --start=2004/6/29:17 \
         --pass=stdout \
         --proto=6 | \
rwstats --pair-top-threshold=1000
```

This produces the result shown in Figure 4.

This example shows that there was one SYN flood that occurred during the hour that was examined. We can look at the flows in detail by using the command:

```
INPUT SIZE: 127393 records
SOURCE IP/DEST IP PAIRS: Top 10 of 95825 unique
src_ip_addr     |dest_ip_addr     |  num_pairs| %_of_input|   cumul_%
10.100.1.10     |241.21.208.42    |         99|  0.077712%| 0.077712%
10.10.10.10     |241.22.97.159    |         52|  0.040819%| 0.118531%
10.110.100.10   |241.240.17.204   |         22|  0.017269%| 0.135800%
10.120.100.10   |241.241.17.204   |         21|  0.016484%| 0.152285%
10.10.1.1       |241.242.1.51     |         14|  0.010990%| 0.163274%
10.130.100.100  |241.243.200.199  |         10|  0.007850%| 0.171124%
10.10.1.10      |241.244.187.97   |         10|  0.007850%| 0.178974%
10.140.10.100   |241.245.231.202  |          8|  0.006280%| 0.185254%
10.150.100.100  |241.23.240.114   |          5|  0.003925%| 0.189178%
10.150.100.100  |241.24.128.179   |          5|  0.003925%| 0.193103%
```

Figure 3: Output from rwstats --pair-topn=10.

```
INPUT SIZE: 4477703 records
SOURCE IP/DEST IP PAIRS: Top 30 of 3953344 unique
src_ip_addr     |dest_ip_addr     |  num_pairs| %_of_input|   cumul_%
241.240.220.58  |10.100.100.100   |      20893|  0.466601%| 0.466601%
```

Figure 4: Output from rwstats --pair-top-threshold=1000.

```
rwfilter --saddr=241.240.220.58 \
         --daddr=10.100.100.100 \
         --start=2004/6/29:17 \
         --pass=stdout | \
rwsort --field=9 | \
rwcut --field=3-8 | less
```

This command filters on the particular source and destination IP address of interest for the one hour, followed by sorting the records based on the start time for the flow. A sample of the results from this command are:

sPort	dPort	pro	packets	bytes	flags
54237	17299	6	1	60	S
54232	38318	6	1	60	S
54235	62020	6	1	60	S
54238	46925	6	1	60	S
54239	23970	6	1	60	S
54240	3568	6	1	60	S
54233	43740	6	1	60	S
54228	14472	6	1	60	S
54241	17440	6	1	60	S

This is an unusual set of traffic in that it appears that the attacker was flooding a particular machine, rather than a specific service. It is also unusual for the TCP SYN packet to contain 60 bytes. Further, it appears that the DoS was directed against only high-numbered ports.

To determine if there was any variation in the protocol, packets, bytes or flags, we run:

```
rwfilter --saddr=241.240.220.58 \
    --daddr=10.100.100.100 \
    --start=2004/6/29:17 \
    --pass=stdout | \
rwuniq --field=6
```

In this case, we are looking at how many different numbers of packets (field=6) appear in the set of flows. By varying the field value, we can also examine bytes, flags, and protocol. In this case we found that all of the flows were 1-packet TCP SYN flows consisting of 60 bytes. By choosing field=4 for destination port, and then piping the result through sort and wc, we found that 13915 unique ports were targeted, with no port being hit more than three times.

Infected Machines

Another example usage comes from tracking the MyDoom worm in late January, 2004. This worm spread via an email attachment that created a backdoor on ports 3127-3198. After the release of this worm, scanning for this backdoor increased significantly. To see the number of flows caused by this scanning in 10-minute intervals (indicated by --bin-size=600, for 600 seconds) over the 26-27 January 2004, we use the commands:

```
rwfilter --start-date=2004/1/26:00 \
    --end-date=2004/1/27:23 \
    --dport=3127 --proto=6 \
    --type=in --pass=stdout | \
rwcount --bin-size=600
```

Note that we use only the incoming non-web data, and not the web data. This is because port 3127 can be chosen as an ephemeral port for web connections, which is benign traffic that we want to exclude. The rwfilter command processed 354,559,695 records, generating output that consisted of only 104,376 records to be processed by rwcount. In this case, we use the default binning of rwcount, which is to put the flow in the bin based on its start time, regardless of the elapsed time of the flow. For example, if a flow consisted of 10,000 bytes over 20 minutes, then all 10,000 bytes would be counted in the first 10-minute bin (based on start time), rather than 5000 bytes in the first 10-minute bin, and 5000 bytes in the second 10-minute bin. One of the options provided with rwcount will split the bytes and packets evenly over the time period covered by the record. This could result in fractional value (and hence we provide two digits after the decimal place for precision in the output). A snapshot of some of the result is provided in Figure 5.

Two interesting events occur in this data. The first is a sudden jump in the number of bytes transferred,

Date	Records	Bytes	Packets
01/26/2004 07:40:00	5.00	4508.00	21.00
01/26/2004 07:50:00	5.00	3468.00	63.00
01/26/2004 08:00:00	6.00	47078833.00	36509.00
01/26/2004 08:10:00	9.00	93215.00	123.00
...			
01/27/2004 20:40:00	9.00	6152.00	63.00
01/27/2004 20:50:00	9240.00	786257.00	14840.00
01/27/2004 21:00:00	1010.00	90580.00	1683.00
01/27/2004 21:10:00	34569.00	2788388.00	53526.00
01/27/2004 21:20:00	28810.00	2326538.00	44585.00
01/27/2004 21:30:00	9039.00	735054.00	14112.00
01/27/2004 21:40:00	7.00	15842.00	101.00

Figure 5: Output from filtering on destination port 3127 and then looking at the number of bytes, packets and flows in 10 minute intervals.

```
rwfilter --stime=2004/1/26:08:00:00- 2004/1/26:08:10:00 dport.3127 \
--pass=stdout | rwcut
```

Figure 6: Filtering on ten minute interval.

even though the number of flows remained constant. Drilling down to investigate further, we filter on the 10 minute interval and then print the resulting records; see Figure 6. There was one flow in this time period that accounted for the majority of bytes, which was a transfer from port 119, which contains the network news protocol, but also the Happy99 trojan [12]. However, by going to the source IP address, we find that it is a news server, and so this traffic is likely legitimate.

The second interesting event is the sudden jump in the number of records, which likely represents scanning activity against our network. We can determine which source IP addresses had the most flows associated with them by using the command in Figure 7. This command prints all IP addresses that had more than 10 flows in the one hour time period. There were only two IP addresses that met this criterion, one of which had 12 flows, and the second of which had 82,639. It is therefore likely that this second IP was performing a scan of our network.

It is interesting to determine if there was any traffic that was returned to the scanning IP address. To do this, we filter all outgoing traffic on the scanning IP address as a destination (here, we represent this IP as 241.2.3.4); see Figure 8. If there had been multiple scanning IP addresses, we could perform the same operation by creating an ipset first and then filtering on this set. We now have a file that contains all of the return traffic to the (potential) scanner(s).

Examining this file more closely, we find 2658 flows. We are particularly interested in flows that do not consist of only a RST-ACK. To determine if any flows meet this criteria, we can filter on all flows that contain just a RST-ACK, and then look at those flows that fail this filter:

```
rwfilter --rst=1 --ack=1 \
         --urg=0 --psh=0 \
         --syn=0 --fin=0 \
         --proto=6 response.scanners \
         --fail=stdout | \
rwcut --fields=1-8 | less
```

There are only 10 records that fail this query. Fortunately, all 10 records were ICMP error messages, and so we can conclude that no internal machines had the trojan running.

Comparison to Related Work

The work that is the most closely related to this work is that of OSU FlowTools, developed by Fullmer and Romig [3]. The OSU FlowTools is a great toolkit, and we had initially investigated using it. However, it was not capable of processing the amount of data we had in the time required, nor did it compress information sufficiently to minimize disk space. While FlowTools has continued to be developed over the past two years (with the latest release appearing to be December 2003), increasing the efficiency of processing flows or storing to diskspace has not been a priority. Indeed, for the majority of networks, OSU FlowTools is more than sufficient. However, our needs correspond to providing analysis tools for a large ISP, where long-term trending as well as short-term security analysis were requirements. We therefore developed our own flow packing system with performance and disk space minimization as design goals. To maintain information on 1.5 billion flows requires approximately 30 Gb of disk space. Additional space savings can be obtained through compression. (Saving this information as raw NetFlow records requires approximately 67 GB.) These flows can be processed (via rwfilter) in 21 minutes on a Sun 4800.

Many of the basic tools we provide are the same as in the OSU FlowTools, such as the ability to filter flows on ports or addresses, or to perform some level of statistical analysis. However, OSU FlowTools relies on Unix utilities for items such as sorting and uniq'ing, while we have developed utilities that perform these operations on the raw data. By using these customized utilities, the performance increases significantly. For example, we can sort 45,433,086 records in five minutes, instead of 11.5 minutes required to sort the ASCII output.

One of the capabilities that we do provide, that appears to be missing in OSU FlowTools, is that of ipsets. This provides a user with the ability to generate any arbitrary list of IP addresses (such as a list of known scanners, or known hostile hosts, or key internal servers) and use this list in an efficient manner as a filter option. This functionality has proven to be particularly useful for security analysis. For example, earlier we showed how to use ipsets to store a list of scanning IP addresses, which we can then use to filter outgoing data to search for SYN-ACK responses to these scans, which might indicate potential compromises. Assume that there were 1000 scanners in whom we were interested (rather than just the two in the example). OSU FlowTools would require the user to create an acl file with the IP addresses of interest in it in order to achieve the desired filtering. In contrast, we can generate the ipset of interest from the first rwfilter, and use this to then filter the outgoing data. Again, our

```
rwfilter --stime=2004/1/27:20:40:00- 2004/1/27:21:40:00 dport.3127 \
--pass=stdout | rwaddrcount --print-rec --rec-min=10
```

Figure 7: Finding IP addresses with most flows.

```
rwfilter --start-date=2004/1/27:20 --end-date=2004/1/27:21
--class=out --type=in --daddr=241.2.3.4 --pass=response.scanners
```

Figure 8: Filter by scanning IP address.

approach has been optimized for performance (using a tree rather than a linear list), so that there is no reduction in filtering speed as the number of IPs in the ipset increases. Another example would be a "bad list," containing the IP addresses of external hosts who are known to have exhibited malicious activity in the past. The bad list can be represented as an ipset, and then the incoming data can be filtered on the bad list IPs as sources. Similarly the outgoing data can be filtered with the bad list as destinations. If we receive a bad list from another site that we wish to merge with our own, we need only do an rwset-union to combine the two sets into one.

In addition, we provide the ability to extend the filtering capabilities of rwfilter through the use of dynamic libraries. Using this approach, administrators can program their own queries for cases where their query is too complex for the current filtering options. One example of where a dynamic library is useful is in examining flow traffic for particular patterns of activity. For example, one sign of a successful buffer overflow is that a source first contacted a server *S* on port *P*, and that this was then followed with a subsequent communication from the source to server *S* but on port *R* (e.g., the first flow represents 241.9.9.9 → 10.8.8.8:80, and the second flow is 241.9.9.9 → 10.8.8.8:5483). Every time a flow showed a connection with more than one 40-byte packet to port 80 on some destination, then the information could be stored in a hash table with the source and destination IPs as the key. This hash table would be checked each time a flow was encountered that did not meet the above condition. If such a match was found, then the entry in the hash table would be marked. Once all records were processed, all marked entries in the hash table could be printed. To the best of our knowledge, none of the other flow tools provide this capability.

Another useful capability that is provided by the SiLK Suite is rwfileinfo, which allows a user to determine information about a packed file. What is particularly useful about this command is that it will return the arguments that were provided to rwfilter in order to generate the data file.

Conclusions and Future Work

We have presented a new suite of tools for saving and analyzing NetFlow data. The tools provided were built with network security analysis in mind, and can be easily extended by a knowledgeable C programmer through both the creation of new tools and the incorporation of dynamic libraries. The tools were specifically designed for use on very large and very busy networks, and so had fast execution and minimal disk space usage as design requirements.

We have completed the collection system and provided basic analysis tools. We now intend to supplement these capabilities by providing tools that allow traffic descriptions. One example of this is bags, which will be provided in an upcoming open source release. Bags are similar to ipsets, except that rather than using a single bit to indicate if an IP address has been seen, it provides a 32-bit counter that counts the number of flows seen to/from each IP address. This allows a user to ask questions such as "What IP addresses saw only one flow in the past hour?" and "How many IP addresses saw more than 10,000 flows in the past day?" Tools such as these will allow an administrator to characterize their network in cases where they might not otherwise have the authority or insight to do so (e.g., such as in the cases of large ISPs). Bags will be extended in a future release to be even more generic, counting any type of "volume" characteristic (e.g., flows, bytes, packets).

In addition, we intend to provide the ability to perform stateful queries. For example, we are working on an rwmatch tool, which will match flows from two sets of data based on a specific attribute. For example, we could filter all incoming flows to a particular port (e.g., TCP 135) into one file, generating the ipset for the sources at the same time. We could then use this ipset to filter all outgoing traffic to an ephemeral port (> 1024), and save the resulting data. rwmatch would use the two output files, and match on the IP addresses (destination in one direction matching with source in the other direction). This would provide an aggregated flow record indicating the traffic in both directions in a single record. This would allow an administrator to see all relevant data at once (e.g., the number of bytes and packets in each direction, for example), rather than needing to manually eyeball two different data files.

The current tool suite has already been in operational use at a large site for over a year and is currently used by several different organizations. Additionally, extensions have been coded that have been used in security publications. Two papers have been written that make use of this tool set, with some custom-coded extensions. McHugh [6] provides a good explanation of how to use the functionality of IP sets, along with the bags extension, for security analysis. Collins and Reiter [2] have used the SiLK tool set in performing an analysis of denial of service (DoS) traffic-filtering approaches.

Acknowledgments

The authors would like to acknowledge the helpful suggestions from Marc Kellner, Jim McCurley, Tom Longstaff, Tim Shimeall and John McHugh from the CERT Network Situational Awareness Group, as well as the many analysts at the client site. We would also like to thank our shepherd, David Hoffman, for his helpful and constructive suggestions.

Author Information

Carrie Gates has five years of system administration experience, starting at a small not-for-profit

organization and finishing with three years as the System Manager for the Computer Science Faculty at Dalhousie University. She left this position in 2001 to pursue a Ph.D., specializing in network security. She is currently a Visiting Scientist with the CERT Network Situational Awareness program at Carnegie Mellon University, where she is completing her dissertation research. She can be reached at cgates@cert.org .

Michael Collins is a full-time employee of CERT/NETSA where he focuses on network traffic analysis and the study of large systems. Before working for CERT, he worked for several years in the n-dim group, studying engineering design and system reliability. He graduated with a B.S. in Physics from CMU in 1997, and a MS in Electrical Engineering in 2001. He is a candidate for a Ph.D. in Electrical Engineering. He can be reached at mcollins@cert.org .

Michael Duggan graduated with a Bachelors degree in Electrical and Computer Engineering from Carnegie Mellon University in 1996, after which he worked as a software developer. He joined the CERT Network Situational Awareness program in 2003, where he worked on the SiLK Suite of tools, concentrating on the collection system.

Andrew Kompanek is a member of the Network Situational Awareness Team at the Software Engineering Institute at Carnegie Mellon University. Prior to joining the SEI, he was a member of the research staff in the School of Computer Science at Carnegie Mellon, a principal at a startup, and a partner in a software development consultancy. He can be reached at ajk@cert.org .

Mark Thomas is a programmer and analyst for the Network Situational Awareness team, part of the Networked Systems Survivability Program at the Software Engineering Institute (SEI). Mark holds a Ph.D. and a MS in Chemical Engineering from Carnegie Mellon University and a BS in Chemical Engineering from West Virginia Institute of Technology.

References

[1] CAIDA, *cflowd: Traffic Flow Analysis Tool*, http://www.caida.org/tools/measurement/cflowd , 2004.

[2] Collins, Michael and Michael Reiter, "An empirical analysis of target-resident DoS filters," *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 103-114, May 9-12, 2004.

[3] Fullmer, Mark and Steve Romig, "The OSU flow-tools package and Cisco Netflow logs," *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, pages 291-303, Usenix Organization, December 3-8, 2000.

[4] Internet Assigned Numbers Authority (IANA), *Protocol numbers*, http://www.iana.org/assignments/protocol-numbers , 2004.

[5] Kompanek, Drew and Mark Thomas, *SiLK Analysis Suite*, http://sourceforge.net/projects/silktools/ , 2003.

[6] McHugh, John, "Sets, Bags and Rock and Roll," *Proceedings of the Ninth European Symposium on Research in Computer Security*, September 13-15, 2004.

[7] "The Honeynet Project," *Know Your Enemy*, Addison-Wesley, 2002.

[8] QoSient, LLC, *Argus: Network Audit Record Generation and Utilization System*, http://www.qosient.com/argus/ , 2004.

[9] Symantec, *W32.Korgo.F*, http://securityresponse.symantec.com/avcenter/venc/data/w32.korgo.f.html , 2004.

[10] Symantec, *W32.Sasser.B.Worm*, http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.b.worm.html , 2004.

[11] Cisco Systems, *Cisco CNS NetFlow Collection Engine*, http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/products_user_guide_chapter09186a00801ed569.html , 2004.

[12] Treachery Unlimited, *Port Lookup Search Results*, http://www.treachery.net/tools/ports/lookup.cgi , 2004.

[13] Williamson, Matthew M., "Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code," *18th Annual Computer Security Applications Conference*, December 9-13, 2002.

Real-time Log File Analysis Using the Simple Event Correlator (SEC)

John P. Rouillard – University of Massachusetts at Boston

ABSTRACT

Log analysis is an important way to keep track of computers and networks. The use of automated analysis always results in false reports, however these can be minimized by proper specification of recognition criteria. Current analysis approaches fail to provide sufficient support for the recognizing the temporal component of log analysis. Temporal recognition of event sequences fall into distinct patterns that can be used to reduce false alerts and improve the efficiency of response to problems. This paper discusses these patterns while describing the rationale behind and implementation of a ruleset created at the CS department of the University of Massachusetts at Boston for SEC – the Simple Event Correlation program.

Introduction

With today's restricted IT budgets, we are all trying to do more with less. One of the more time consuming, and therefore neglected, tasks is the monitoring of log files for problems. Failure to identify and resolve these problems quickly leads to downtime and loss of productivity. Log files can be verbose with errors hidden among the various status events indicating normal operation. For a human, finding errors among the routine events can be difficult, time consuming, boring, and very prone to error. This is exacerbated when aggregating events, using a mechanism such as syslog, due to the intermingling of events from different hosts that can submerge patterns in the event streams.

Many monitoring solutions rely on summarizing the log files for the previous days logs. This is very useful for accounting and statistics gathering. Sadly, if the goal is problem determination and resolution then reviewing these events the day after they are generated is less helpful. Systems administrators cannot proactively resolve or quickly respond to problems unless they are aware that there is a problem. It is not useful to find out in the next morning's summary that a primary NFS server was reporting problems five minutes before it went down. The sysadmin staff needs to discover these problems while there is still time to fix the problem and avert a catastrophic loss of service.

The operation of computers and computer networks evolves over time and requires a solution to log file analysis that address this temporal nature. This paper describes some of the current issues in log analysis and presents the rationale behind an analysis rule set developed at the Computer Science Department at the University of Massachusetts at Boston. This ruleset is implemented for the Simple Event Correlator (SEC), which is a Perl based tool designed to perform analysis of plain text logs.

Current Approaches

There are many programs that try to isolate error events by automatically condensing or eliminating routine log entries. In this paper, I do not consider interactive analysis tools like **MieLog** [Takada02]. I separate automatic analysis tools into offline or batch monitoring and on-line or real-time monitoring.

Offline Monitoring

Offline solutions include: **logwatch** [logwatch], **SLAPS-2** [SLAPS-2], or **Addamark LMS** [Sah02]. Batch solutions have to be invoked on a regular basis to analyze logs. They can be run once a day, or many times an hour. Offline tools are useful for isolating events for further analysis by real time reporting tools. In addition they provide statistics that allow the system administrator to identify the highest event sources for remedial action. However, offline tools do not provide the ability to provide automatic reactions to problems. Adam Sah in discussing the **Addamark LMS** [Sah02, p. 130] claims that real-time analysis is not required because a human being, with slow reaction times, is involved in solving the problem. I disagree with this claim. While it is true that initially a human is required to identify, isolate and solve the problem, once it has been identified, it is a candidate for being automatically addressed or solved. If a human would simply restart apache when a particular sequence of events occur, why not have the computer automatically restart apache instead? Automatic problem responses coupled with administrative practices can provide a longer window before the impact of the problem is felt. An "out of disk space" condition can be addressed by removing buffer files placed in the file system for this purpose. This buys the system administrator a longer response window in which to locate the cause of the disk full condition minimizing the impact to the computing environment.

Most offline tools do not provide explicit support for analyzing the log entries with respect to the time they were received. While they could be extended to try to parse timestamps from the log messages, this is difficult in general, especially with multiple log files and multiple machines, as ordering the events requires

normalizing the time for all log messages to the same timezone. Performing analysis on log files that do not have timestamps eliminates the ability of these batch tools to perform analysis by time. Solutions such as the **Addamark LMS** [Sah02] parse and record the generation time, but the lack of real-time event-driven, as opposed to polled, triggers reduces its utility.

Online Monitoring

Online solutions include: **logsurfer** [logsurfer], **logsurfer+** [logsurfer+], **swatch** [swatch, Hansen1993], **2swatch** [2swatch], **SHARP** [Bing00], **ruleCore** [ruleCore], **LoGS** [LoGS] and **SEC** [SEC]. All of these programs run continuously watching one or more log files, or receiving input from some other program.

Swatch is one of the better known tools. **Swatch** provides support for ignoring duplicate events and for changing rules based on the time of arrival. However, **swatch's** configuration language does not provide the ability to relate arbitrary events in time. Also, it lacks the ability to activate/deactivate rules based on the existence of other events other than suppressing duplicate events using its throttle action.

Logsurfer dynamically changes its rules based on events or time. This provides much of the flexibility needed to relate events. However, I found its syntax difficult to use (similar to the earliest 1.x version of SEC) and I never could get complex correlations across multiple applications to work properly. The dynamic nature of the rules made debugging difficult. I was never able to come up with a clean, understandable, and reliable method of performing counting operations without resorting to external programs. Using SEC, I have been able to perform all of the operations I implemented in **logsurfer** with much less confusion.

LoGS is an analysis program written in Lisp that is still maturing. While other programs create their own configuration language, **LoGS's** rules are also written in Lisp. This provides more flexibility in designing rules than SEC, but may require too much programming experience on the part of the rule designers. I believe this reduces the likelihood of its widespread deployment. However, it is an exciting addition to the tools for log analysis research.

The Simple Event Correlator (SEC) by Risto Vaarandi uses static rules unlike **logsurfer**, but provides higher level correlation operations such as explicit pair matching and counting operations. These correlation operations respond to a triggering event and persist for some amount of time until they timeout, or the conditions of the correlation are met. SEC also provides a mechanism for aggregating events and modifying rule application based on the responses to prior events. Although it does not have the dynamic rule creation of **logsurfer**, I have been able to easily generate rules in SEC that provide the same functionality as my **logsurfer** rules.

Filter In vs. Filter Out

Should rules be defined to report just recognized errors, or should routine traffic be eliminated from the logs and the residue reported? There appear to be people who advocate using one strategy over the other.

I claim that both approaches need to be used and in more or less equal parts. I am aware of systems that are monitored for only known problems. This seems risky as it is more likely an unknown problem will sneak up and bite the unwary systems administrator. However, very specific error recognition is needed when using automatic responses to ensure that the best solution is chosen. Why restart Apache if killing a stuck CGI program will solve the problem?

Filtering out normal event traffic and reporting the residue allows the system administrator to find signatures of new unexpected problems with the system. Defining "normal traffic" in such a way that we can be sure its routine is tricky especially if the filtering program does not have support for the temporal component of event analysis.

Event Modeling

Modeling normal or abnormal events requires the ability to fully specify every aspect of the event. This includes recognizing the content of the event as well as its relationship to other events in time. With this ability, we can recognize a composite or correlated event that is synthesized from one or more primitive events. Normal activity is usually defined by these composite events. For example a normal activity may be expressed as:

> 'sendmail -q' is run once an hour by root at 31 minutes after the hour. It must take less than one minute to complete.

```
> CMD: /usr/lib/sendmail -q
> root 25453 c Sun May 23 03:31:00 2004
< root 25453 c Sun May 23 03:31:00 2004
```

Figure 1: Events indicating normal activity for a scheduled cron job.

This activity is shown by the cron log entries in Figure 1 and requires the following analysis operations:

- Find the sendmail CMD line verifying its arrival time is around 31 minutes after the hour. If the line does not come in, send a warning.
- The next line always indicates the user, process id and start time. Make sure that this line indicates that the command was run by root.
- The time between the CMD line arrival and the final line must be less than one minute. Because other events may occur between the start and end entries for the job, we recognize the last line by its use of the unique number from the second field of the second line.

This simple example shows how a tool can analyze the event log in time. Tools that do not not allow

the specification of events in the temporal realm as well as in the textual/content space can suffer from the following problems:

- matching the right event at the wrong time. This could be caused by an inadvertent edit of the cron file, or a clock skew on the source or analyzing host.
- not noticing that the event took too long to run.
- not noticing that the event failed to complete at all.

Temporal Relationships

The cron example mentions one type of temporal restriction that I call a schedule restriction. Schedule restrictions are defined by working on a defined (although potentially complex) schedule. Typical schedule restrictions include: every hour at 31 minutes past the hour, Tuesday morning at 10 a.m., every weekday morning between 1 and 3 a.m.

In addition to schedule restrictions, event recognition requires accounting for inter-event timing. The events may be from a single source such as the sequence of events generated by a system reboot. The statement that the first to last event in a boot sequence should complete in five minutes is an inter-event timing restriction. Also, events may arise from multiple sources. Multi-source inter-event timing restrictions might include multiple routers sending an SNMP authentication trap in five minutes, or excessive "connection denied" events spread across multiple hosts and multiple ports indicating a port scan of the network.

These temporal relationships can be explicit within a correlation rule: specifying a time window for counting the number of events, suppressing events for a specified time after an initial event. The timing relationship can also be implicit when one event triggers the search for subsequent events.

Event Threading

Analysis of a single event often fails to provide a complete picture of the incident. In the example above, reporting only the final cron event is not as useful as reporting all three events when trying to diagnose a cause. Lack of proper grouping can lead to underestimating the severity of the events. Consider the following scenario:

1. A user logs in using ssh from a location that s/he has never logged in from before.
2. The ssh login was done using public key authentication.
3. The ssh session tries to open port 512 on the server. It is denied.
4. Somebody tries to run a program called "crackme" that tries to execute code on the stack.
5. The user logs out.

Looking at this sequence implies that somebody broke in and tried to execute an unsuccessful attempt to gain root privileges. However, in looking at individual events, it is easy to miss the connections. Taken in isolation, each event could be easily dismissed, or even filtered out of the reports. Reporting them as discrete events, as many analysis tools do, may even contribute to an increased chance of missing the pattern. Taken together they indicate a problem that needs to be investigated. A log analysis tool needs to provide some way to link these disparate messages from different programs into a single thread that paints a picture of the complete incident.

Missing Events

Log analysis programs must be able to detect missing log events [Finke2002]. These missing events are critical errors since they indicate a departure from normal operation that can result in a many problems. For example, cron reports the daily log rotation at 12:01 a.m. If this job is not done (say, because cron crashed), it is better to notice the failure immediately rather than three months later when the partition with the log files fills up.

The problem with detecting missing events is that log monitoring is – by its nature – an event-driven operation: if there is no event, there is no operation. The log analysis tool should provide some mechanism for detecting a missing event. One of the simpler ways to handle this problem is to generate an event or action on a regular basis to look for a missing event. An event-driven mechanism can be created using external tools such as cron to synthesize events, but I fail to see a mechanism that the log analysis tool can use to detect the failure of the external tool to generate these events.

Handling False Positives/False Negatives

A false negative occurs when an event that indicates a problem is not reported. A false positive results when a benign event is reported as a problem. False negatives impact the computing environment by failing to detect a problem. False positives must be investigated and impact the person(s) maintaining the computing environment. A false positive also has another danger. It can lead to the "boy who cried wolf" syndrome, causing a true positive to be ignored as a false positive.

Two scenarios for generating false negatives are mentioned above. Both are caused by incorrectly specifying the conditions under which the events are considered routine.

False positives are another problem caused by insufficiently specifying the conditions under which the event is a problem. In either case, it may not be possible to fully specify the problem conditions because:

- Not all of the conditions are known.
- Some conditions are not able to be monitored and cannot be added to the model.

It may be possible to find correlative conditions that occur to provide a higher degree of discrimination in the model. These correlative events can be use to change the application of the rules that cause the false positive to inhibit the false report.

To reduce these false positives and false negatives, the analysis program needs to have some way of generating and receiving these correlative events.

While it is impossible to eliminate all false reports, by proper specification of event parameters, false reports can be greatly reduced.

Single vs. Multiple Line Events

Programs can spread their error reports across multiple lines in a logfile. Recognizing a problem in these circumstances requires the ability to scan not just a single line, but a series of lines as a single instance. The series of lines can be treated as individual events, but key pieces of information needed to trigger a response or recognize an event sequence may occur on multiple lines. Consider the cron example of Figure 1: the first two lines provide the information needed to determine that it is an entry for sendmail started by root, and the process id is used in discovering the matching end event. Handling this multi-line event as multiple single line events complicates the rules for recognizing the events.

Multi-line error messages seem to be more prevalent in application and device logs that do not use the Unix standard syslog reporting method, but some syslog versions split long syslog messages into multiple parts when they store them in the logfile. Fortunately, when I have seen this happen, the log lines always occur adjacent to one other without any intervening events from other sources. This allows recognition provided that the split does not occur in the middle of a field of interest.

With syslog and other log aggregation tools, a single multi-line message can be distorted by the injection of messages from other sources. The logs from applications that produce multi-line messages should be directed to their own log file so that they are not distorted. Then a separate SEC process can analyze the log file and create single line events that are passed to a parent SEC for global correlation. This is similar to the method used by **Addamark** [Sah02].

Although keeping the log streams separate simplifies some log analysis tasks, it prevents the recognition of conditions that affect multiple event streams. Although SEC provides a mechanism for identifying the source of an event, performing event recognition across streams requires that the event streams be merged.

SEC Correlation Idioms and Strategies

This section describes particular event scenarios that I have seen in my analysis of logs. It demonstrates idioms for SEC that model and recognize these scenarios.

SEC Primer

A basic knowledge of SEC's configuration language is required to understand the rules presented below. There are nine basic rule types. I break them into two groups: basic and complex rules. Basic rules types perform actions and do not start an active correlation operation that persists in time. These basic types are described in the SEC man page as:

- **Suppress**: suppress matching input event (used to keep the event from being matched by later rules).
- **Single**: match input event and immediately execute an action that is specified by rule.
- **Calendar**: execute an action at specific times using a cron like syntax.

Complex rules start a multi-part operation that exists for some time after the initial event. The simplest example is a SingleWithSuppress rule. It triggers on an event and remains active for some time to suppress further occurrences of the triggering event. A Pair rule recognizes a triggering event and initiates a search for a second (paired) event. It reduces two separate but linked events to a single event pair. The complex types are described in the SEC man page as:

- **SingleWithScript**: match input event and depending on the exit value of an external script, execute an action.
- **SingleWithSuppress**: match input event and execute an action immediately, but ignore following matching events for the next T seconds.
- **Pair**: match input event, execute the first action immediately, and ignore following matching events until some other input event arrives (within an optional time window T). On arrival of the second event execute the second action.
- **PairWithWindow**: match input event and wait for T seconds for another input event to arrive. If that event is not observed within a given time window, execute the first action. If the event arrives on time, execute the second action.
- **SingleWithThreshold**: count matching input events during T seconds and if given threshold is exceeded, execute an action and ignore all matching events during rest of the time window.
- **SingleWith2Thresholds**: count matching input events during T1 seconds and if a given threshold is exceeded, execute an action. Now start to count matching events again and if their number per T2 seconds drops below second threshold, execute another action.

SEC rules start with a type keyword and continue to the next type keyword. In the example rules below, '...' is used to take the place of keywords that are not needed for the example, they do not span rules. The order of the keywords is unimportant in a rule definition.

SEC uses Perl regular expressions to parse and recognize events. Data is extracted from events by using subexpressions in the Perl regular expression. The extracted data is assigned to numeric variables $1, $2, ..., $N where N is the number of subexpressions in the Perl regular expression. The numeric variable

$0 is the entire event. For example, applying the regular expression "([A-z]*): test number ([0-9]*)" to the event "HostOne: test number 34" will assign $1 the value "HostOne", $2 the value "34", and $0 will be assigned the entire event line.

Because complex rule types create ongoing correlation operations, a single rule can spawn many active correlation operations. Using the regular expression above, we could have one correlation that counted the number of events for Host and another separate correlation that counted events for HostTwo. Both counting correlations would be formed from the same rule, but by extracting data from the event the two correlations become separate entities.

This data allows the creation of unique contexts, correlation descriptions and coupled patterns linked to the originating event. We will explore these items in more detail later. Remember that when applying a rule, the regular expression or pattern is always applied first regardless of the ordering of the keywords. As a result, references to $1, $2, ..., $N anywhere else in the rule refer to the data extracted by the regular expression.

SEC provides a flow control and data storage mechanism called contexts. As a flow control mechanism, contexts allow rules to influence the application of other rules. Contexts have the following features:

- Contexts are dynamically created and often named using data extracted from an event to make names unique.
- Contexts have a defined lifetime that may be infinite. This lifetime can be increased or decreased as a result of rules or timeouts.
- Multiple contexts can exist at any one time.
- A context can execute actions when its lifetime expires.
- Contexts can be deleted without executing any end-of-lifetime actions.
- Rules (and the correlations they spawn) can use boolean expressions involving contexts to determine if they should apply. Existing contexts return a true value; non-existent contexts return a false value. If the boolean expression is true, the rule will execute, if false the rule will not execute (be suppressed).

In addition to a flow control mechanism, contexts also serve as storage areas for data. This data can be events, parts of events or arbitrary strings. All contexts have an associated data store. In this paper, the word "context" is used for both the flow control entity and its associated data store. When a context is deleted, its associated data store is also deleted. Contexts are most often used to gather related events, for example login and logout events for a user. These contexts can be reported to the system administrator if certain conditions are detected (e.g., the user tried to perform a su during the login session).

The above description might seem to imply that a single context has a single data store; this is not always the case. Multiple contexts can share the same data store using the alias mechanism. This allows events from different streams to be gathered together for reporting or further analysis. The ability to extract data from an event and linking the context by name to that event provides a mechanism for combining multiple event streams into a single context that can be reported. For example, if I extract the process ID 345 from syslog events, I can create a context called: process_345 and add all of the syslog events with the same PID to that event. If I now link the context process_346 to the process_345 context, I can add all of the syslog events with the pid 346 to the same context (data store). So now the process_345/process_346 context contains all of the syslog events from both processes.

In the paper, I use the term 'session.' A session is simply a record of events of interest. In general these events will be stored in one or more contexts. If ssh errors are of interest, a session will record all the ssh events into a context (technically a context data store that may be known by multiple names/aliases) and report that context. If tracing the identities that a user assumes during a login is needed, a different series of data is recorded in a context (data store): the initial ssh connection information is recorded, the login event, the su event as the user tries to go from one user ID to another.

The rest of the elements of SEC rules will be presented as needed by the examples.

Responding To Or Filtering Single Events

The majority of items that we deal with in processing a log file are single items that we have to either discard or act upon. Discardable events are the typical noise where the problem is either not fixable, for example a failing reverse DNS lookups on remote domains from tcp wrappers, or are valueless information that we wish to discard.

Discardable events can be handled using the suppress rule. Figure 2 is an example of such a rule.

```
type=suppress
desc=ignore non-specific paper problem \
     report since prior events have \
     given us all we need.
ptype=regexp
pattern=. printer: paper problem$
```

Figure 2: A suppress rule that is used to ignore a noise event sent during a printer error. *Note: SEC example rules are reformatted/split for readability. They may or may not work exactly as presented.*

Since all of my rule sets report anything that is not handled, we want to explicitly ignore all noise lines to prevent them from making it to the default "report everything" rule.

This is a good time to look at the basic anatomy of a SEC rule. All rules start with a type option as

described earlier. All rules have a desc option that documents the rule's purpose. For the complex correlation rules, the description is used differentiate between correlation operations derived from a single rule. We will see an example of this when we look at the horizontal port scan detection rules.

Most rules have a pattern option that is applied to the event depending on the ptype option. The pattern can be a regular expression, a substring, or a truth value (TRUE or FALSE). The ptype option specifies how the pattern option is to be interpreted: a regular expression (regexp), a substring (substr), or a truth value (TValue). It also determines if the pattern is successfully applied if it matches the event match (regexp/substr), or does not match (nregexp/nsubstr) the event. For TValue type patterns, TRUE matches any event (successful application), while FALSE (not successfully applied) does not match any input event. If the pattern does not successfully apply, the rule is skipped and the next rule in the configuration file is applied.

A number can be added to the end of any of the nregexp, regexp, substr, or nsubstr values to make the pattern match across that many lines. So a ptype value of regexp2 would apply the pattern across two lines of input.

By default when an event triggers a rule, the event is *not compared* against other rules in the same file. This can be changed on a per rule basis by using the continue option.[1]

After single event suppression, the next basic rule type is the single rule. This is used to take action when a particular event is received. Actionable events can interact with other higher level correlation events: adding the event to a storage area (context), changing existing contexts to activate or deactivate other rules, activating a command to deal with the event, or just reporting the event. Figure 3 is an example of a single rule that will generate a warning if the printer is offline from an unknown cause.

In Figure 3 we see two more rule options: context and action. The context option is a boolean expression of contexts that further constrains the rule.

When processing the event

```
lj2.cs.umb.edu: printer: Report Printer
                          Offline if needed
```

the single rule in Figure 3 checks to see if the pattern applies successfully. In this case the pattern matches the event, but if the Report_Printer_lj2.cs.umb.edu_Offline context does not exist, then the actions will not be executed. The context Report_Printer_lj2.cs.umb.edu_Offline is deleted by other rules in the ruleset (not shown) if a more exact diagnosis of the cause is detected. This suppresses the default (and incorrect) report of the problem.

The action option specifies the actions to take when the rule fires. In this case it writes the message printer lj2.cs.umb.edu offline, unknown cause to standard output (specified by the file name "-") . Then it deletes the context Report_Printer_lj2.cs.umb.edu_Offline since it is no longer needed.

There are many potential actions, including:

- creating, deleting, and performing other operations on contexts
- invoking external programs
- piping data or current contexts to external programs
- resetting active correlations
- evaluating Perl mini-programs
- setting and using variables
- creating new events
- running child processes and using the output from the child as a new event stream.

We will discuss and use many of these actions later in this paper.

Scheduling Events With Finer Granularity

Part of modeling normal system activity includes accounting for scheduled activities that create events. For example, a scheduled weekly reboot is not worth reporting if the reboot occurs during the scheduled window, however it is worth reporting if it occurs at any other time.

For this we use the calendar rule. It allows the reader to schedule and execute actions on a cron like schedule. In place of the ptype and pattern options it has a time option that has five cron-like fields. It is wonderful for executing actions or starting intervals on a minute boundary. Sometimes we need to start intervals with resolution of a second rather than a minute.

Figure 4 shows a mechanism for generating a window that starts 15 seconds after the minute and lasts for 30 seconds. The key is to create two contexts and use both of them in the rules that should be active (or inactive) only during the given window. One context wait_for_window expires to begin the timed interval. The window context expires marking the end of the interval. Creating an event on a non-minute boundary is trivial once the reader learns that the event command has a built in delay mechanism.

[1]Note: continue is not supported for the suppress rule type.

```
type=single
continue=dontcont
desc = Report Printer Offline if needed
ptype=regexp
pattern=^(\w._-]+): printer: Report Printer Offline if needed
context = Report_Printer_$1_Offline
action = write - "printer $1 offline, unknown cause" ; \
         delete Report_Printer_$1_Offline
```

Figure 3: A single command that writes a warning message and deletes a context that determines if it should execute.

Triggering events generated by calendar rules or by expiring contexts can execute actions, define intervals, trigger rules or pass messages between rules. Triggering events are used extensively to detect missing events.

```
type=calendar
time=30 3 * * *
desc=create 30 second window
action=create window 45; \
       create wait_for_window 15

type=single
...
context=window && !wait_for_window
```

Figure 4: A mechanism for creating an timed interval that starts on a non-minute boundary.

Detecting Missing Events

The ability to generate arbitrary events and windows with arbitrary start and stop times is useful when detecting missing events. The rules in Figure 5 report a problem if a 'sendmail -q' command is not run by root near 31 minutes after the hour. Because of natural variance in the schedule, I expect and accept a sendmail start event from five seconds before to 10 seconds after the 31st minute.

The event stream from Figure 1 is used as input to the rules. Figure 6 displays the changes that occur while processing the events in time. Contexts are represented by rectangles, the length of the rectangle is the context's lifetime. Upside down triangles represent the arrival of events. Regular triangles represent actions within SEC. The top graph shows the sequence when the sendmail event fails to arrive, while the bottom graph shows the sequence when the sendmail program is run.

The correlation starts when rule 2 (the calendar rule) creating the context sendmail_31_minute that will execute an action (write a message to standard output) when it times out after 70 seconds (near 31 minutes and 10 seconds) ending the interval. The calendar rule creates a second context, sendmail_31_minute_inhibit, that will timeout in 55 seconds (near 30 minutes and 55 seconds) starting the 15 second interval for the arrival of the sendmail event. Looking at the top graph in Figure 7, we see the creation of the two contexts on the second and third lines. No event arrives within the 15 second window, so the sendmail_31_minute expires and executes the "write" action. The bottom graph shows what happens if the sendmail event is detected. Rule 1 is triggered by the sendmail event occurring in the 15

```
# rule 1: detect the sendmail event
type = single
desc = sendmail has run, don't report it as failed
ptype = regexp2
pattern = ^\> CMD: /usr/lib/sendmail -q.*\n\> root ([0-9]+) c .*
context = sendmail_31_minute && ! sendmail_31_minute_inhibit
action = delete sendmail_31_minute

# rule 2: define the time window and prep to report a missing event
type = calendar
desc = Start searching for sendmail invocation at 31 past hour
time=30 * * * *
action = create sendmail_31_minute 70 write - \
              Sendmail failed to run detected at %t; \
         create sendmail_31_minute_inhibit 55
```

Figure 5: Rules to detect a missed execution of a sendmail process at the appointed time.

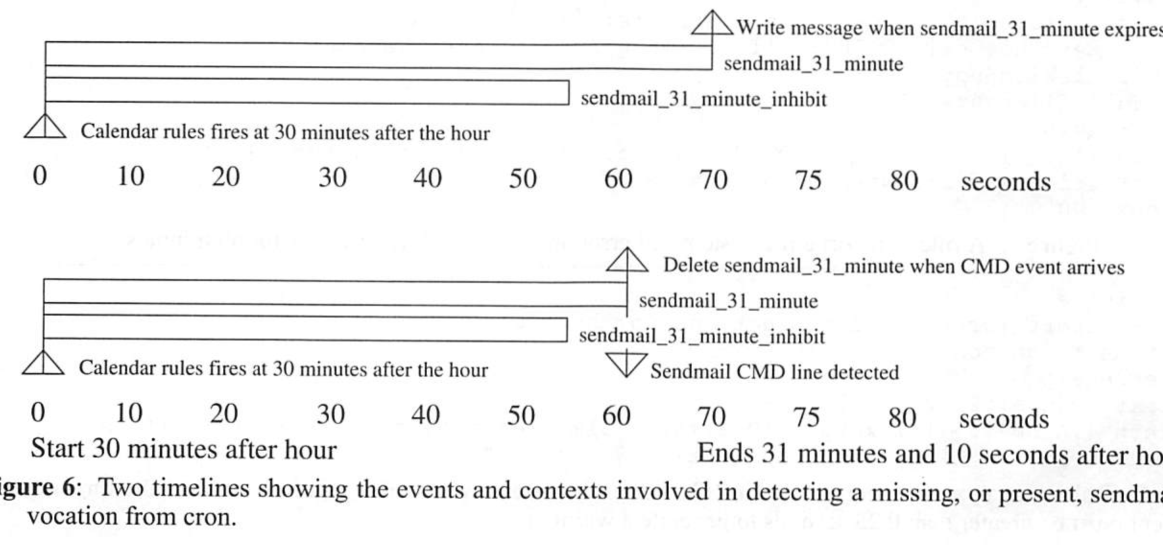


Figure 6: Two timelines showing the events and contexts involved in detecting a missing, or present, sendmail invocation from cron.

seconds window and deletes the sendmail_31_minute context. The deletion also prevents the "write" action associated with the context from being executed.

Note that the boolean context of rule 1 prevents its execution if the sendmail event were to occur less than five seconds before the 31st minute since ! sendmail_31_minute_inhibit is false because sendmail_31_minute_inhibit exists and is therefore true. If the sendmail event occurs after 31 minutes and 10 seconds, the context is again false since sendmail_31_minute does not exist, and is false.

The example rules use the write action to report a problem. In a real ruleset, the systems administrator could use the SEC shellcmd action to invoke logger(1) to generate a syslog event to be forwarded to a central syslog server. This event would be found by SEC running on the syslog master. The rule matching the event could notify the administrator via email, pager, wall(1) or send a trap to an NMS like HPOV or Nagios. Besides reporting, the event could be further processed with a threshold rule that would try to restart cron as soon as two or more "missed sendmail events" events are reported, and report a problem only if a third consecutive "missed sendmail event" arrived.

Repeat Elimination/Compression

I have dealt with real-time log file reporters that generated 300 emails when a partition filled up overnight. There must be a method to condense or de-duplicate repeated events to provide a better picture of a problem, and reduce the number of messages spamming the administrators.

The SingleWithSuppress rule fills this de-duplication need. To handle file system full errors, the rule in Figure 7 is used.

This rule reports that the filesystem is full when it receives its first event. It then suppresses the event message for the next hour. Note that the desc keyword includes the filesystem and hostname ($2 and $1 respectively). This makes the correlation operation that is generated from the rule unique so that a disk full condition on the same host for the filesystem /mount/fs2 will generate an error event if it occurs five minutes after the /mount/sd0f event. If the filesystem was not included in the desc option, then only one alert for a full filesystem would be generated regardless of how many filesystems actually filled up during the hour.

Report on Analysis of Event Contents

Unlike most other programs, SEC allows the reader to extract and analyze data contained within an event. One simple example is the rule that analyzes NTP time adjustments. I consider any clock with less than 1/4 a second difference from the NTP controlled time sources to be within a normal range. Figure 8 shows the rules that are applied to analyze the xntpd time adjustment events. We extract the value of the time change from the step messages. This value is assigned to the variable $1. The context expression executes a Perl mini-program to see if the absolute value of the change is larger than the threshold of 0.25 seconds. If it is, the context is satisfied and the rule's actions fire.

The context expression uses a mechanism to run arbitrary Perl code. It then uses the result of the expression to determine if the rule should fire. It can be used to match networks after applying a netmask, perform calculations with fields of the event or other tasks to properly analyze the events.

Detect Identical Events Occurring Across Multiple Hosts

A single incident can affect multiple hosts. Detecting a series of identical events on multiple hosts provides a measure of the scope of the problem. The problem can be an NFS server failure affecting only one host that does not need to be paged out in the middle of the night, or it may affect 100 hosts, which requires recovery procedures to occur immediately.

```
# Example:
#   Apr 13 15:08:52 host4.example.org ufs: [ID 845546 \
#       kern.notice] NOTICE: alloc: /mount/sd0f: file system full
type=SingleWithSuppress
desc=Full filesystem $2 on $1
ptype=regexp
pattern=(\w._-]+) ufs: \[.* NOTICE: alloc: (\w/._-]+): file system full
action= write - filesystem $2 on host $1 full
window=3600
```

Figure 7: A rule to report a file system full error and suppress further errors for 60 minutes.

```
type=single
desc = report large xntpd corrections for host $1
continue = dontcont
ptype=regexp
context= =(abs($2) > 0.25)
pattern=([A-z0-9._-]+) xntpd\[[0-9]+\]:.*time reset \(step\) ([-]?[0-9.]+) s
action= write - "large xntpd correction($2) on $1"
```

Figure 8: Rule to analyze time corrections in NTP time adjustment events. The absolute value of the time adjustment must be greater than 0.25 seconds to generate a warning.

Other problems such as time synchronization, or detection of port scans also fall into this realm.

One typical example of this rule is to detect horizontal port scans. The rules in Figure 9 identify a horizontal port scan as three or more connection denied events from different server hosts within five minutes from a particular external host or network. So 20 connections to different ports on the same host would not result in the detection of a horizontal scan. In the example, I assume that the hosts are equipped with TCP wrappers that report denied connections. The set of rules in Figure 9 implements the detection of a horizontal port scan by counting unique client host/server host combinations. A timeline of these three rules is shown in Figure 10.

The key to understanding these rules is to realize that the description field is used to match events with correlation operations. When rule 1, the threshold correlation rule, sees the first rejected connection from 192.168.1.1 to 10.1.2.3, it generates a Count denied events from 192.168.1.1 correlation. The next time a deny for 192.168.1.1 arrives, it will be tested by rule 1, the description field generated from this new event will match an ongoing correlation threshold operation and it will be considered part of the Count denied events from 192.168.1.1 threshold correlation. If a rejection event for the source 193.1.1.1 arrives, the generated description field will not match an active threshold correlation, so a new correlation operation will be started with the description Count denied events from

```
# Example input:
# May 10 13:52:13 cyber TCPD-Event cyber:127.6.7.1:3424:sshd deny \
#     badguy.example.com:192.268.15.45 user unknown
#  Variable = description (value from example above)
#  $3  = server ip address (127.6.7.1)
#  $5  = daemon or service connected to on server (sshd)
#  $8  = ip address of client (attacking) machine (192.268.15.45)
#  $9  = 1st quad of client host ip address (192)
#  $10  = 2nd quad of client host ip address (6)
#  $11 = 3rd quad of client host ip address (7)
#  $12 = 4th quad of client host ip address (1)
# Rule 1: Perform the counting of unique destinations by client host/net
type = SingleWithThreshold
desc = Count denied events from $8
continue = takenext
ptype = regexp
pattern = ^(.*) TCPD-Event ([A-z0-9_.]*):([0-9.]*):([0-9]*):([^ ]*) (deny) \
            ([^:]*):(([0-9]*)\.([0-9]*)\.([0-9]*)\.([0-9]*)) user (.*)
action = report conn_deny_from_$8 /bin/cat >> report_log
context = ! seen_connection_from_$8_to_$3
thresh = 3
window = 300
## Rule 2: Insert a rule to capture synthesized network tcpd events.
type=single
...
pattern = ^(.*) TCPD-Event ([A-z0-9_.]*):([0-9.]*):([0-9]*):([^ ]*) (deny) \
            ([^:]*):(([0-9]*)\.([0-9]*)\.([0-9]*)\.([0-9]*)) user (.*) net$
action=none

# Rule 3: Generate network counting rules and maintain contexts
type = single
desc = maintain counting contexts for deny service $5 from $8 event
continue = takenext
ptype = regexp
pattern = ^(.*) TCPD-Event ([A-z0-9_.]*):([0-9.]*):([0-9]*):([^ ]*) (deny) \
            ([^:]*):(([0-9]*)\.([0-9]*)\.([0-9]*)\.([0-9]*)) user (.*)
context = ! seen_connection_from_$8_to_$3
action = create seen_connection_from_$8_to_$3 300; \
         add conn_deny_from_$8 $0 ; \
         event 0 $1 TCPD-Event $2:$3:$4:$5 $6 $7:$9.$10.$11.0 user $13 net; \
         event 0 $1 TCPD-Event $2:$3:$4:$5 $6 $7:$9.$10.0.0 user $13 net; \
         event 0 $1 TCPD-Event $2:$3:$4:$5 $6 $7:$9.0.0.0 user $13 net; \
         add conn_deny_from_$9.$10.$11.0 $0 ; \
         add conn_deny_from_$9.$10.0.0 $0 ; \
         add conn_deny_from_$9.0.0.0 $0
```

Figure 9: Rules to detect horizontal port scans defined by connections to 3 different server hosts from the same client host within 5 minutes. Note: patterns are split for readability. This is not valid for sec input.

source2. Figure 10 shows a correlation operation from start to finish. First the event E1 reports a denial from host 192.168.1.1 to connect/scan 10.1.2.3. The correlation operation Count denied events from 192.168.1.1 is started by rule 1, rule 2 is skipped because the pattern does not match, and rule 3 creates the 5-minute-long context seen_connection_from_192.168.1.1_to_10.1.2.3 that is used to filter arriving event to make sure that only unique events are counted. The rest of rule 3's actions will be discussed later.

The count for rule 1, the threshold correlation operation, is incremented only if the seen_connection_from_192.168.1.1_to_10.1.2.3 context does not exist. When the E1/2 (event 1 number 2) arrives, this context still exists and all the rules ignore the event. When E2/1 arrives, it triggers rule 1 and rule 3 creating the appropriate context and incrementing the threshold operation's count.

When five minutes have passed since E1/1's arrival and the threshold rule has not been triggered by the arrival of three events, the start of the threshold rule is moved to the second event that it counted, and the count is decremented by 1. This occurs because the threshold rule uses a sliding window by default. When events 3/1 and 4/1 arrive, they are counted by the shifted threshold correlation operation started by rule 1. With the arrival of E2/1, E3/1, and E4/1, three events have occurred within five minutes and a horizontal port scan is detected. As a result, the action reporting the context conn_deny_from_192.168.1.1 is executed and the events counted during the correlation operation (maintained by the add action of rule 3) are reported to the file report_log.

Rule 2 and the final actions of rule 3 allow detection of horizontal port scans even if they come from different hosts such as: 192.168.3.1, 192.168.1.1, and 192.168.7.1. If each of these hosts scans a different host on the 10 network, it will be detected as a horizontal scan from the 192.168.0.0 network. This is done by creating three events replacing the real source address with a corresponding network address. One event is created for each class A, B and C network that the original host could belong to: 192.168.1.0, 192.168.0.0, and 192.0.0.0. The response to these synthesized events are not shown in Figure 10, but they start a parallel series of correlation operations and contexts using the network address of the client in place of 192.168.1.1.

Vertical scans can use the same framework with the following changes:

- the filtering context needs to include port numbers so that only unique client host/server host/port triples are counted by the threshold rule.
- the description of rule 1 to include the server host IP so that it only counts connections to a specific server host.

This will count the number of unique server ports that are accessed on the server from the client host.

In general, using rules 1 and 3, you can count unique occurrences of a value or group of values. The context used to link the rules must include the unique values in its name. The description used in rule 1 will not include the unique values and will create a bucket in which the events will be counted. In the horizontal port scan case, case, my bucket was any connection from the same client host. The unique value was the server IP address connected to by the the client host. In detecting a vertical port scan, the value is the number of unique ports connected to while the bucket is the client/server host pair.

These two changes allow the counting ruleset to count the number of unique occurrences of the parameter that is present in the filtering rule, but missing from the rule 1 description (the bucket), e.g., if the context

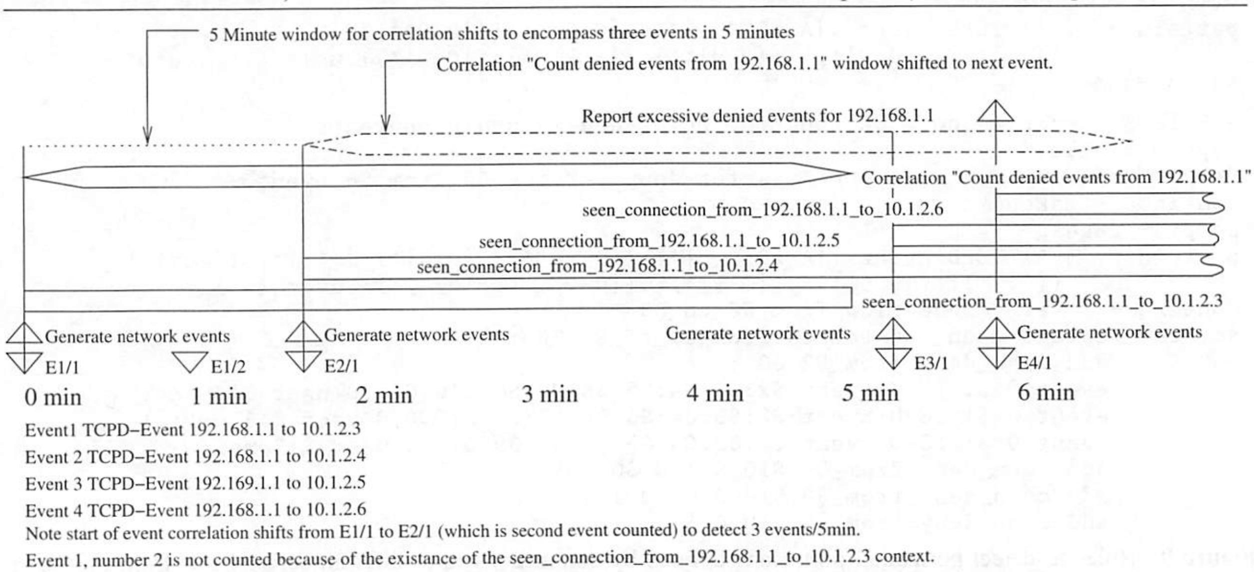


Figure 10: Timeline showing the application of rules to detect horizontal port scans.

specifies serverhost, clienthost, serverport and rule 1 specifies clienthost and serverhost in its description, then the rules above implement counting of unique ports for a given clienthost and serverhost. The rules as presented above specified clienthost and serverhost, rule 1 specified the clienthost, so the ruleset counted unique serverhost's for a given clienthost.

Other counting methods can also be implemented using mixtures of the vertical and horizontal counting methods.

While I implemented a "pure" SEC solution, the ability to use Perl functions and data structured from SEC rules provides other solutions [Vaarandi7_2003] to this problem.

Creating Threads of Events from Multiple Sources

Many thread recognition operations involve using one of the pair type rules. Pair rules allow identification of a future (child) event by searching for identifying information taken from the present (parent) event. This provides the ability to stitch a thread through various events by providing a series of pair rules.

There are three times when you need to trigger an action with pair rules:

1. Take action upon receipt of the parent event
2. Take action upon receipt of the child event
3. Take action after some time when the child event has not been received (expiration of the pair rule).

The Pair rule provides actions for triggers 1 and 2. The PairWithWindow rule provides actions for triggers 2 and 3. None of the currently existing pair rules provides a mechanism for taking actions on all three triggers. Figure 11 shows a way to make up for this limitation by using a context that expires when the pair rule is due to be deleted. Since triggers 2 and trigger 3 are mutually exclusive, part of trigger 2's action is to delete the context that implements the action for to trigger three.

I have used this method for triggering an automatic repair action upon receipt of the first event. The arrival of the second event indicated that the repair worked. If the second event failed to arrive, an alert would be sent when the context timed out. Also, I have triggered additional data gathering scripts from the first event. The second event in this case reported the event and additional data when the the end of the additional data was seen. If the additional data did not arrive on time, I wanted the event to be reported.

This mechanism can replace combinations of PairWithWindow and Single rules. It simplifies the rules by eliminating duplicate information, such as patterns, that must be kept up to date in both rules.

Correlating Across Processes

One of more difficult correlation tasks involves creating a session made up of events from multiple processes.

Figure 12 shows a ruleset that sets up a link between parent and child ssh processes. Its application is show in Figure 13.

When a connection to ssh occurs, the parent process, running as root, reports the authentication events and generates information about a user's login. After the authentication process, a child sshd is spawned that is responsible for other operations including port forwarding and logout (disconnect) events. The ruleset in Figure 12 captures all of the events generated by the parent or child ssh process. This includes errors generated by the parent and child ssh processes.

A session starts with the initial network connection to the parent sshd and ends with a connection closed event from the child sshd. I accumulate all events from both processes into a single context. I also have rules (not shown in the example) to report the entire context when unexpected events occur.

The tricky part is accumulating the events from both processes into a single context. The connection between the event streams is provided by a tie event that encompasses unique identifying elements from both event streams and thus ties together the two streams into a single stream.

Each ssh process has its own unique event stream stored in the context session_log_<hostname>_<pid>. There is a Single rule, omitted for brevity, that accumulates ssh events into this context. When the tie event is seen, it provides the link between the parent sshd session_log context and the child session_log context. The data from the two contexts is merged and the two context names (with the parent and child pid's) are assigned to the same underlying context. Hence the child's session_log_<hostname>_<child pid> context and the parent's session_log_<hostname>_<parent pid> contexts refer to the same data. After the contexts are linked, actions using either the child context name or the parent context name

```
type=pair
...
action = write - rule triggered ; \
         create take_action_on_pair_expiration 60 (write - rule expired)
...
pattern2=
action2 = write - pattern 2 seen ; \
          delete take_action_on_pair_expiration
...
window=60
```

Figure 11: A method to take an action on all three trigger points in a pair rule.

operate on the same underlying context. Reporting or adding to the context using one of the linked names acts the same regardless of which name is used.

In Figure 14 the first event E1 triggers rule 1 from Figure 13, the PairWithWindow rule, to recognize the start of the session. The second half of rule 1 looks for a tie event for the following 60 seconds. There may be many tie events, but there should be only one tie event that contains the the pid of the parent sshd. Since we have that stored in $2, we use it in pattern2. The start of session event is passed onto additional rules (not shown) by setting the continue option on rule 1 to takenext. These additional rules record the events in the session_log context identified by system and pid, as in the session_log_example. org_10240 context of Figure 14.

If the tie event is not found within 60 seconds, the session_log_example.org_10240 context is reported. However, if the tie event is found as in Figure 13, then a number of other operations occur. The tie event is generated by a script that is run by the child sshd. Therefore it is possible for the child sshd to generate events before the tie event is created. Because of the default rule that adds events to the session_log_example. org_10245, additional work must be done when the tie event arrives to preserve the data in the child's session_log. The second part of rule 1 in Figure 12 copies child's session_log context into the variable %b. The child's session_log is then deleted and aliased to the parent session_log. The %2 variable is the value of $2 from the first pattern, the parent process's PID. After pattern2 is applied, the parent PID is referenced as %2 because $2 is now the second subexpression of pattern2. Next the data copied from the child log is injected into the event stream to allow re-analysis and reporting using the combined parent and child context.

The last action for the tie event is to alias the login username stored in the context session_log_owner_<hostname>_<parent pid> to a similar context under the child pid. Then any rule that analyzes a child event can obtain the login name by referencing the alias context. Rule 2 in Figure 12 handles the login event and creates the context session_log_owner_<hostname>_<parent pid> where it stores the login name for use by the other rules in the ruleset. Rule 2 also stores the login event in the session_log context.

The last rule is very simple. It detects the "close connection" (logout) event and deletes the contexts created during the session. The delivery of event N (EN) in Figure 13 causes deletion of contexts. Deleting an aliased context deletes the context data store as well as all the names pointing to the context data store.

```
# rule 1 - recognize the start if an ssh session,
#          and link parent and child event contexts.
type=PairWithWindow
continue=takenext
desc=Recognize ssh session start for $1[$2]
ptype=regexp
pattern=([A-Za-z0-9._-]+) sshd\[([0-9]+)\]: \[[^]]+\] Connection from ([0-9.]+) \
        port [0-9]+
action=report session_log_$1_$2 /bin/cat
desc2=Link parent and child contexts
ptype2=regexp
pattern2=([A-Za-z0-9._-]+) [A-z0-9]+\[[0-9]+\]: \[[^]]+\] SSHD child process +([0-9]+\
          spawned by $2
action2=copy session_log_$1_$2 %b; \
        delete session_log_$1_$2; \
        alias session_log_$1_%2 session_log_$1_$2; \
        add session_log_$1_$2 $0; \
        event 0 %b; \
        alias session_log_owner_$1_%2 session_log_owner_$1_$2; \
window=60

# rule 2 - recognize login event and save username for later use
type=single
desc=Start login timer
ptype=regexp
pattern=([A-Za-z0-9._-]+) sshd\[([0-9]+)\]: \[[^]]+\] Accepted \
    (publickey|password) for ([A-z0-9_-]+) from [0-9.]+ port [0-9]+ (.*)
action=add session_log_$1_$2 $0; add session_log_owner_$1_$2 $4

# rule 3 - handle logout
type=single
desc=Recognize ssh session end
ptype=regexp
pattern=([A-Za-z0-9._-]+) sshd\[([0-9]+)\]: \[[^]]+\] Closing connection to ([0-9.]+)
action= delete session_log_$1_$2; delete session_log_owner_$1_$2
```

Figure 12: Accumulating output from ssh into a single context.

Rule 3 uses the child PID to delete session_log_example.org_10245 and session_log_owner_example.org_10245, which cleans up all four context names (two from the parent PID and two from the child) and both context data stores.

This mechanism can be used for correlating any series of events and passing information between the rules that comprise an analysis mechanism. The trick is to find suitable tie events to allow the thread to be followed. The tie event must contains unique elements found in the events streams that are to be tied together. In the ssh correlation I create a tie event using the pid's of the parent and child events. Every child event includes the PID of the child sshd so that I can easily construct the context name that points to the combined context data store. For the ssh correlation, I create the tie event by running shell commands using the sshrc mechanism and use the logger(1) command to inject the tie event into the data stream. This creates the possibility that the tie event arrives after events from the child process. It would make the correlation easier if I modified the sshd code to provide this tie event since this would generate the events in the correct order for correlation.

Having the events arriving in the wrong order for cross correlation is a problem that is not easily remedied. I suppress reporting of the child events while waiting for the tie event (not shown). Then once the tie event is received, the child events are resubmitted for correlation. This is troublesome and error prone and is an area that warrants further investigation.

Strategies to Improve Performance

One major issue with real-time analysis and notification is the load imposed on the system by the analysis tool and the rate of event processing. The rules can be restructured to reduce the computational load. In other cases the rule analysis load can be distributed across multiple systems or across multiple processes to reduce the load on the system or improve event throughput for particular event streams.

The example rule set from UMB utilizes a number of performance enhancing techniques. Originally these techniques were implemented in a locally modified version of SEC. As of SEC version 2.2.4, the last of the performance improvements has been implemented in the core code.

Rule Construction

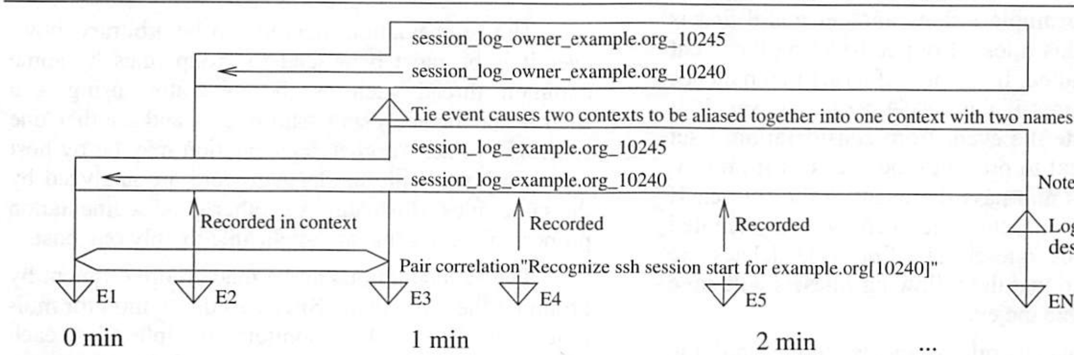
For SEC, construction of the rules file(s) plays a large role in improving performance. In SEC, the majority of computation time is occupied with recognizing events using Perl regular expressions. Optimizing these regular expressions to reduce the amount of time needed to apply them improves performance.

However, understanding that SEC applies each rule sequentially allows the reader to put the most often matched rules first in the sequence. Putting the most frequently used rules first reduces the search time needed to find an applicable rule. Sending a USR1 signal to SEC causes it to dump its internal state showing all active contexts, current buffers, and other information including the number of times each rule has been matched. This information is very useful in efficiently restructuring a ruleset

Using rule segmentation to reduce the number of rules that must be scanned before a match is found proves the biggest gains for the least amount of work.

Rule Segmentation

In August 2003, I developed a method of using SEC's multiple configuration file mechanism to prune the number of rules that SEC would have to test before finding a matching rule.



E1 – Initial parent event: example.org sshd[10240]: [ID 800047 auth.info] Connection from 192.168.0.1 port 3500
E2 – Parent event: example.org sshd[10240]: [ID 800047 auth.info] Accepted publickey for rouilj from 192.168.0.1 port 3500 ssh2
E3 – Tie event: example.org rouilj[10248]: [ID 702911 auth.notice] SSHD child process 10245 spawned by 10240
E4 – Child event: example.org sshd[10245]: [ID 800047 auth.info] error: connect_to 127.0.0.1 port2401 failed
E5 – Child event: example.org sshd[10245]: [ID 800047 auth.info] bind: Cannot assign requested address
EN – Logout child event: example.org sshd[10245]: [ID 800047auth.info] Closing connection to 192.168.0.1

Note 1: creation of session_log_example.org_10240 context in response to E1 is done by a catchall rule that is not shown in the ruleset.
Note 2: alias of two contexts is shown by the large box labeled with both context names.

Figure 13: The application of the ssh ruleset showing the key events in establishing the link between parent and child processes.

This mechanism provides a limited branching facility within SEC's ruleset. A single criteria filtering rule is shown in Figure 14.

```
type=suppress
continue=dontcont
ptype=NRegExp
pattern=^[ABCD]
desc=guard for abcd rules

type=single
continue=dontcont
ptype=TValue
pattern=true
desc=guard for events handled by other \
    ruleset files
action=logonly
context = [handled]
type=single
continue=takenext
ptype=TValue
pattern=true
desc=report handled
action=create handled
<rules here>

type=single
ptype=TValue
pattern=true
desc=Guess we didn't handle this event \
    after all
action=delete handled
```

Figure 14: A sample rule set to allow events to be filtered and prevented from matching other rules in the file.

This rule depends on the Nregexp pattern type. This causes the rule to match if the pattern *does not* match. The pattern is crafted to filter out events that can not possibly be acted upon by the other rules in the file. In this example I show another guard that is used to prevent this ruleset from considering the event if it has been handled. It consists of a rule that matches all events[2] and fires if the handle context is set. If it does not eliminate the event from consideration, I set the handled context to prevent other rulesets from processing the event and pass the event to the ruleset. If the final rule triggers, then the event was not handled by any rule in the ruleset. The final rule deletes the handled context so that the following rulesets will have a chance to analyze the event.

Note that this last rule is repeated in the final rule file to be applied. There it resets the handled context so that the next event will be properly processed by the rulesets.

In addition to a single regexp, multiple patterns can be applied and if any of them select the event, the event will be passed through the rest of the rules in the file. A rule chain to accept an event based on multiple patterns is shown in Figure 15. The multiple filter criteria can be set up to accept/reject the event using complex boolean expressions so that the event must match some patterns, but not other patterns.

```
type= single
desc= Accept event if match2 is seen.
continue= takenext
ptype= regexp
pattern= match2
action= create accept_rule

type= single
desc= Accept event if match3 is seen.
continue= takenext
ptype= regexp
pattern= match3
action= create accept_rule

type= single
desc= Skipping ruleset because neither \
    match2 or match3 were seen.
ptype= TValue
pattern= true
context= ! accept_rule
action= logonly

type= single
desc= Cleaning up accept_rule context \
    since it has served its purpose.
continue=takenext
ptype= TValue
pattern= true
context= accept_rule
action= delete accept_rule; logonly

<other rules here>
```

Figure 15: A ruleset to filter the input event against multiple criteria. The words "match2" or "match3" must be seen in the input event to be processed by the other rules.

The segmentation method can be arbitrary, however it is be most beneficial to group rules by some common thread such as the generator, using a a file/ruleset for analyzing sshd events and another one for xntp events. Another segmentation may be by host type. So hosts with similar hardware are analyzed by the same rules. Hostname is another good segmentation property for rules that are applicable to only one host.

The segmentation can be made more efficient by grouping the input using SEC's ability to monitor multiple files. When SEC monitors multiple files, each file can have a context associated with it. While processing a line from the file, the context is set. For example, reading a line from /var/adm/messages may set the adm_messages context, while reading a line from /var/log/syslog would set the log_syslog context and clear the adm_messages context. This allows segmentation of rules by source file. Offloading the work of grouping to an external application such as syslog-ng provides the ability to group the events not only by facility and level as in classic syslog, but also by other

[2]The TValue ptype is only available in SEC 2.2.5 and newer. Before that use regexp with a pattern of / ^ . ? /.

parameters including host name, program, or by a matching regular expression. Since syslog-ng operates on the components of a syslog message rather than the entire message, it is expected to be more efficient in segmenting the events than SEC.

Restructuring the rules for a single SEC process using a simple five file segmentation based on the first letter of the event using an 1800 rule ruleset increased throughput by a factor of three. On a fully optimized ruleset of 50 example rules, running on a SunBlade 150 (128 MB of memory, 650 Mhz), I have seen rates exceeding 300 lines/sec with less than 40% processor utilization. In tests run under the Cygwin environment on Microsoft windows 2000, 40 rules produced a throughput of 115 log entries per second. This single file path of 40 rules is roughly equivalent to a segmented ruleset of 17 files with 20 rules each for a total of 340 rules, with events equally distributed across the rulesets.

Note that these throughput numbers depend on the event distribution, the length of the events etc. Your mileage may vary.

Parallelization of Rule Processing

In addition to optimizing the rules, multiple SEC processes can be run, feeding their composite events to a parent SEC. SEC can watch multiple input streams. It merges all these streams into a single stream for analysis. This merging can interfere with recognition of multi-line events as well as acting to increase the size of an event queue, slowing down the effective throughput rate of a single event stream. Running a child SEC process on an event stream allows faster response to that stream.

SEC's `spawn` action creates a process and creates an event from every line emitted by the child process. The events from these child processes are placed on the front of the event queue for faster processing.

These features allow the creation of a hierarchy of SEC processes to process multiple rules files. This reduces the burden on the parent SEC process by distributing the total number of rules across different processes. In addition, it simplifies the creation of rules when multi-line events must be considered, by preventing the events from being distorted by the injection of other events in the middle of the multi-line event.

SEC is not threaded, so use of concurrent processes is the way to make SEC utilize multiprocessor systems. However, even on uniprocessor systems, it seems to provide better throughput by reducing the mean number of rules that SEC has to try before finding a match.

Distribution Across Nodes

SEC has no built-in mechanism for distributing or receiving events with other hosts. However, one can be crafted using the ideas from the last two sections. Although this has not been tested, it is expected to provide a significant performance improvement.

The basic idea is to have the parent SEC process use ssh to spawn child SEC processes on different nodes. These nodes have rules files that handle a portion of the event stream. The logging mechanisms are set up to split the event streams to the nodes so that each node has to work on only a portion of the event stream. Even if the logs are not split across nodes, the reduced number of rules on each node is expected to allow greater throughput.

This can be used in a cluster to allow each host to process its own event streams and report composite events to the parent SEC process for cross-machine correlation operations.

Limitations

Like any tool, SEC is not without its limitations. The serial nature of applying SEC's rules limits its throughput. Some form of tree-structured mechanism for specifying the rules would allow faster application. One idea that struck me as interesting is the use of ripple-down rulesets for event correlation [Clark2000] that could simply the creation and maintenance or rulesets as well as speed up execution of complex correlation operations.

As can be seen above, a number of idioms consist of mating a single rule to a more complex correlation rule to receive the desired result. This makes it easy to get lost in the interactions of more complex rulesets. I think more research into commonly used idioms, and the generation of new correlation operations to support these idioms will improve the readability and maintainability of the correlation rules.

The power provided by the use of Perl regular expressions is tempered by the inability to treat the event as a series of fields rather than a single entity. For example, I would prefer to parse the event line into a series of named fields, and use the presence, absence and content of those fields to make the decisions on what rules were executed. I think it would be more efficient and less error prone to come up with a standard form for the event messages and allow SEC to tie pattern matches to particular elements of the event rather then match the entire event. However, implementation of the mechanism may have to wait for the "One True Standard for Event Reporting," and I do not believe I will live long enough to see that become a reality.

The choice of Perl as an implementation language is a major plus because it is a more widely known language than C among the audience for the SEC tool this increases the pool of contributers to the application. Also, Perl allows much more rapid development than C. However, using an interpreted language (even one turned into highly optimized bytecode) does cause a slowdown in execution speed compared to native executable.

SEC does not magically parse timestamps. Its timing is based on the arrival time of the event. This

can be a problem in a large network if the travel time cannot be neglected in the event correlation operations.

Future Directions

Refinement of the available rule primitives and actions (e.g., the expire action) is an area for investigation. A number of idioms presented above are more difficult to use than I would like. In some cases these idioms could be made easier by adding new correlation types to the language. In other cases a mechanism for storing and retrieving redundant information (such as regular expressions and timing periods) will simplify the idioms. This may be external using a preprocessor such as filepp or m4, or may be an internal mechanism.

Even though SEC development is ongoing, not every idea needs to be implemented in the core. Using available Perl modules and custom libraries it is possible to create functions and routines to enhance the available functionality without making changes to the SEC core. Developing libraries of add-on routines – as well as standard ways of loading and accessing these routines is an ongoing project. This form of extension permits experimentation without bloating SEC's core.

I would like to see some work done in formalizing the concept of rule segmentation and improving the ability to branch within the rule sets to decrease the time spent searching for applicable rules.

Availability

SEC is available from http://kodu.neti.ee/~risto/sec/.

In addition to the resources at the primary SEC site above, a very good tutorial has been written by Jim Brown [Brown2003] and is available at: http://sixshooter.v6.thrupoint.net/SEC-examples/article.html.

An annotated collection of rules files is available from http://www.cs.umb.edu/~rouilj/sec/sec_rules-1.0.tgz. This expands on the rules covered in this paper and provides the tools for the performance testing as well as a sample sshrc file for the ssh correlation example.

Conclusion

SEC is a very flexible tool that allows many complex correlations to be specified. Many of these complex correlations can be used to model [Prewett] normal and abnormal sequences of events. Precise modeling of events reduces both the false positive and false negative rates easing the burden on system administrators.

The increased accuracy of the model provided by SEC results in faster recognition of problems leading to reduced downtime, less stress and higher more consistent service levels.

This paper has just scratched the surface of SEC's capabilities. Refinements in rule idioms and linkage of SEC to databases are just a few of the future directions for this tool. Just as prior log analysis applications such as **logsurfer** influenced the design and capabilities of SEC, I believe SEC will serve to foster research and push the envelope of current log analysis and event correlation.

Author Biography

John Rouillard is a system administrator whose first Unix experience was on a PDP-11/44 running BSD Unix in 1978. He graduated with a B.S. in Physics from the University of Massachusetts at Boston in 1990. He specializes in automation of sysadmin tasks and as a result is always looking for his next challenging position.

In addition to his system administration job he is also an emergency medical technician. Over the past few years, when not working on an ambulance, he has worked as a planetarium operator, and built test beds for domestic hot water solar heating systems. He has been a member of IEEE since 1987 and can be reached at rouilj@ieee.org.

References

[2swatch] ftp://ftp.sdsc.edu/pub/security/PICS/2swatch/README.

[Brown2003] Brown, Jim, "Working with SEC – the Simple Event Correlator," http://sixshooter.v6.thrupoint.net/SEC-examples/article.html, November 23, 2003.

[Brown5_2004] Brown, Jim, "SEC Logfuscator Project Announcement," simple-evcorr-users mailing list, http://sourceforge.net/mailarchive/forum.php?thread_id=2712448&forum_id=2877, May 3, 2004.

[Clark2000] Clark, Veronica, "To Maintain an Alarm Correlator," Bachelor's thesis, The University of New South Wales, http://www.hermes.net.au/pvb/thesis/, 2000.

[Finke2002] Finke, John, "Process Monitor: Detecting Events That Didn't Happen," *USENIX Systems Administration (LISA 16) Conference Proceedings*, pp. 145-154, USENIX Association, 2002.

[Hansen1993] Hansen, Stephen E. and E. Todd Atkins, "Automated System Monitoring and Notification with Swatch," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp. 145-156, USENIX Association, http://www.usenix.org/publications/library/proceedings/lisa93/hansen.html, November 1993.

[logsurfer] http://www.cert.dfn.de/eng/logsurf/.

[LoGS] Prewett, James E., "Listening to Your Cluster with LoGS," *The Fifth LCI International Conference on Linux Clusters: TheHPC Revolution 2004*, Linux Cluster Institute, http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF04/05-Prewett_J.pdf, May 2004.

[logwatch] Bauer, Kirk, http://www.logwatch.org/.

[logsurfer+] http://www.crypt.gen.nz/logsurfer/.

[NNM] "ManagingYour Network with HP OpenView Network Node Manager," Hewlett-Packard Company, Part number J5323-90000, January 2003.

[Prewett] Prewett, James E., private correspondence, March 2004.

[ruleCore] http://www.rulecore.com .

[Sah02] Sah, Adam, "A New Architecture for Managing Enterprise Log Data," *USENIX Systems Administration (LISA XVI) Conference Proceedings*, pp. 121-132, USENIX Association, November 2002.

[SEC] Vaarandi, Risto, http://kodu.neti.ee/~risto/sec/ .

[SECman] *SimpleEvent Correlator (SEC) manpage*, http://kodu.neti.ee/~risto/sec/sec.pl.html .

[SLAPS-2] *SLAPS-2*, http://www.openchannelfoundation.org/projects/SLAPS-2 .

[SHARP] Bing, Matt and Carl Erickson, "Extending UNIX System Logging with SHARP," *USENIX Systems Administration (LISA XIV) Conference Proceedings*, pp. 101-108, USENIX Association, http://www.usenix.org/publications/library/proceedings/lisa2000/full_papers/bing/bing_html/index.html, December 2000.

[Snare] InterSectAlliance, http://www.intersectalliance.com/projects/SnareWindows/index.html .

[swatch] Atkins, Todd, http://swatch.sourceforge.net/ .

[Takada02] Takada, Tetsuji and Hideki Koike, "MieLog A Highly Interactive Visual Log Browser Using Information Visualization and Statistical Analysis," *USENIX Systems Administration (LISA XVI) Conference Proceedings*, pp. 133-144, USENIX Association, http://www.usenix.org/events/lisa02/tech/takada.html , November 2002.

[Vaarandi7_2003] Vaarandi, Risto, "Re: Is this possible with SEC," simple-evcorr-users mailing list, http://sourceforge.net/mailarchive/forum.php?thread_id=2712448&forum_id=2877 , Jul 4, 2003.

Combining High Level Symptom Descriptions and Low Level State Information for Configuration Fault Diagnosis

Ni Lao – Tsinghua University;
Ji-Rong Wen and Wei-Ying Ma – Microsoft Research Asia
Yi-Min Wang – Microsoft Research

ABSTRACT

Automatic fault diagnosis is an important problem for system management. In this paper, we combine high level symptom descriptions and low level state information to solve the system fault diagnosis problem. We extract state-symptom correlation information from knowledge sources in text format, and then use symptom similarity to rank the candidate system states. We apply the method to Windows Registry problems to help Product Support Service (PSS) engineers. Promising results with two different knowledge sources show the robustness of our method. Finally, we explain why this combination is successful and also discuss its limitations.

Introduction

Configuration management will remain a persistent problem "as long as people change how they want to use the system" [Ande95]. Change and Configuration Management and Support (CCMS) of computer systems with large install bases and large numbers of available third-party software packages have proved to be daunting tasks [LC01]. Jim Gray depicted Trouble-Free System as an important goal of IT research: build a system used by millions of people each day, and yet administered and managed by a single part-time person [Gray03]. To achieve this goal, systems should be self-managing. Redstone and coworkers [RSB03] described a global-scale automated problem diagnosis system that collects problem symptoms from users' desktops, and then automatically searches global databases of problem symptoms and fixes. We address similar problems in a new way in this paper.

People typically use two different strategies to diagnose system faults: symptom-based approach and state-based approach. Nowadays, many systems have knowledge databases of their known problems online (such as [Apple], [BugNet], [MSKB] and [Redhat]). Computer users typically use symptom-based analysis to troubleshoot configuration problems. They describe their problems with words, and use information retrieval tools to find documents containing solutions to the problem. Considering that most customers are not PC experts, their problem descriptions are usually inaccurate and thus using them directly to retrieve relevant documents often yields unsatisfactory results.

At the other extreme, many tools attempt to automate the fault diagnosis task using low level machine states (such as [CKF02] and [Qie03]). Such tools usually provide a language to specify the expected behavior of the system, use monitors to detect system deviation from the rule, and define actions to correct them. For example, Strider [W03] uses various techniques to narrow down the list of candidate root causes, including persistent state differencing, runtime tracing, intersection and statistical ranking. It then uses configuration roll-back [SR00] to fix the problem. Unfortunately, in many cases, the ranking results are not satisfactory. Furthermore, its differencing step and tracing step are not always feasible.

In this study, we combine both high level symptom descriptions and low level state information to solve the configuration fault diagnosis problem. The idea is to extract correlation information between low level states and high level symptom from knowledge sources, and then use symptom similarity to rank the states. We apply the method to Windows Registry problems [Gan04]. Promising results with two different knowledge sources show the robustness of our method. Finally we try to explain why the combination is successful, and discuss its limitations.

System Architecture and User Scenario in PSS

The state-symptom correlation information required for our problem solving technique is extracted from various text-based knowledge sources, such as the Product Support Service (PSS) log and the Microsoft Knowledge Base (KB). The information is then stored in a database called the PC-Genomics Database. Figure 1 illustrates a scenario of how to use PC-Genomics technique for more effective problem troubleshooting.

For example a user named Diana cannot find any fonts in the font dialog box. This is because the

registry keys that list TrueType fonts are damaged. The processes from step (1) through step (7) show how her problem is solved:

- **Step 1**: Diana reports the problem to PSS. She goes to http://support.microsoft.com and describes the problem with a short paragraph.
- **Step 2**: The PSS engineer initially tries to diagnose the problem using the normal method. If this works, go directly to step 7.
- **Step 3**: The state collection and analysis tools are downloaded from the site to Diana's machine.
- **Step 4**: Diana runs Strider to compare bad states and good states in Restore points, and the trace log is also produced.
- **Step 5**: A candidate set containing possible incorrect states is generated from this collected data and sent back to PSS.
- **Step 6**: The candidate set is fed in to PC Genomics database to figure out the root cause of the problem.
- **Step 7**: The generated solution is sent back to the Diana. It could be either a solution script, an executable or related KB articles. In this case, she receives a solution script which deletes the key: key_local_machine\software\microsoft\windows nt\currentversion\fonts .

Extracting State-Symptom Correlation from Knowledge Sources

The PC-Genomics Database

Obtaining the specific data needed for the PC-Genomics Database requires different techniques for different data sources. The digital knowledge sources we use usually consist of articles in free text form. Although we are far from being able to understand the meaning of these articles automatically, we can identify state names and the portion of text which is specifying the problem symptom in these articles. This state-symptom co-occurrence is crucial information to link states with their symptoms. Sometimes the corresponding software name and resolution can also be identified, and the data extracted for the PC-Genomics Database will have the form of Table 1.

The Registry Dictionary

The Windows Registry is the main configuration state store on PCs. It has a tree like structure, and each piece of configuration state is specified by a path name and optionally a value name. Either a path name or a value name prefixed by its path name is called an entry, and there are typically more than 200,000 registry entries on a machine [SR]. To locate registry entries within free format text, we first collected all the registry entries from 50 PCs. They contained 898,546 unique registry entries after name canonicalization (e.g., substituting different user IDs, like "s-1-5-21-..." with the string "UID" in the registry entry path). Then they are used as a "registry dictionary" to help recognize registry entries within free format text.

The Knowledge Sources

The PSS log is an archive of problem-solving cases maintained by Microsoft Corporation. Each case contains the exchanged emails between a customer and a support engineer (see Figure 2). The total PSS

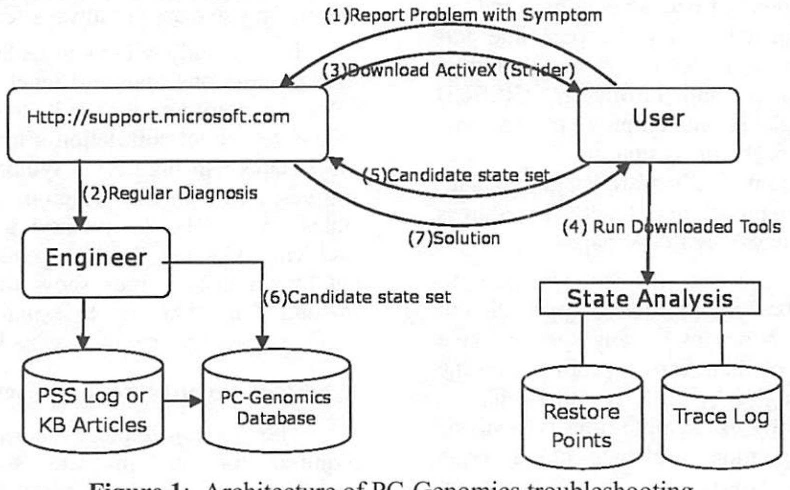


Figure 1: Architecture of PC-Genomics troubleshooting.

ID	State (Registry Key)	Symptom	Software	Solution
1	HKLM\Software\Policies\Microsoft\Messenger\Client	Can not run Windows Messenger...	Messenger	delete PreventRunregister item
2	HKLM\Software\classes\clsid\{f414-a00bb8}\inprocserver32	System Restore GUI is blank...	Windows	1. Go to "Start→Run" 2. . . .

Table 1: Format of the PC genomics database.

log body contains more than 10 million cases. We used 2,311,492 cases in our experiment. These cases cover 15 products within six product families, ranging in time from 3/20/1997 to 5/13/2003 (see Table 2). We combine the *action* and *result* part of the summary as the symptom of a case. If a registry key is referenced in the final mail message of a case, this entry and the case symptom were added into the PC-Genomics Database as a pair. We found that 143,157 of PSS log cases referenced 4,837 unique registry entries, and 1,913 of them are registry values.

-----------------------mail-----------------------
Contact: Dina
System: WIN98 win 98 4.10
Problem: All of my true type fonts have vanished from the font dialog box
-----------------------mail-----------------------
Dear Dina,
There are two things that we need to check. First,...
Sincerely,
Gary
-----------------------mail-----------------------
Dear Gary,
...
Dina
-----------------------mail-----------------------
Dear Dina,
...
I'm going to close your case as successfully solved. Thank you for choosing Microsoft.
Sincerely,
Support Engineer
-----------------------mail-----------------------
SUMMARY
<<Symptom>>
TrueType fonts may not be present in the Fonts folder.
<<Cause>>
The registry key that lists TrueType fonts may be damaged or missing.
<<Resolution>>
Delete the Fonts key and then add it again under:
hkey_local_machine\software\microsoft\windows nt\currentversion

Figure 2: Sample emails in PSS log.

Q329134: Print or Edit Dialog Boxes May Not Appear in Internet Explorer

The information in this article applies to:
Microsoft Internet Explorer version 6 for Windows 2000
Microsoft Internet Explorer 5.5 for Windows 2000 SP 2

SYMPTOMS
When you click Print or Print Preview on the File menu or click Find on the Edit menu in Internet Explorer, the Print and Edit dialog boxes do not appear.

CAUSE
This problem occurs if a corrupted value exists in the registry that may have been written by a third-party installation program.

RESOLUTION
1. Click Start, click Run, type regedit in the Open box, and then click OK.
2. Locate and then click the following registry key:
HKEY_CLASSES_ROOT\CLSID\{00020420-0000-0000-C000-000000000046}\InprocServer32
3. In the right pane, right-click InprocServer32, and then click Delete.

Figure 3: A Sample KB Article.

The Microsoft Knowledge Base [MSKB] contains troubleshooting articles written by experienced engineers (see Figure 3). Our data consist of 142,448 articles ranging from Q10022 to Q332210. The KB articles are written in well formed XML format, so it is easy to parse their symptom section. A registry key found anywhere in the article is added into the PC-Genomics Database with the corresponding symptom. We found that 1,921 of the KB articles reference 996 unique registry entries and 412 of them are registry values.

Rank States Using State-Symptom Correlations

Symptom Similarity

A state-based tool, like Strider, can generate a set of states as candidates for the root cause of a given problem. Our approach matches the symptom of the

Product Name	Product Family Name	Case Count
Office 2000 Win32 EN	Office	88,136
Office SBE 2000 Win32 EN	Office	25,507
Office Pro 2000 Win32 EN	Office	245,429
Office Prem 2000 Win32 EN	Office	149,722
Office Pro XP Win32 EN	Office	80,615
Visual Basic Enter Win 5.0 EN	Visual Basic	17,746
VB Enterprise 6.0 Win32 EN	Visual Basic	38,917
SQL Srvr 7.0 WinNT EN BETA	SQL Server	30,445
SQL Server Ent 7.0 WinNT EN	SQL Server	33,040
Windows Advanced Svr 2000 EN	Windows NT	48,174
Windows Svr 2000 EN	Windows NT	155,892
Exchange Server 5.5 EN	Exchange	251,974
Exchange Svr 2000 EN	Exchange	99,608
Windows XP Home Edition EN	Windows XP	707,908
Windows XP Professional EN	Windows XP	338,379

Table 2: The PSS log used as a knowledge source.

problem with those symptoms of each candidate state in the PC-Genomic database. In this way, we can estimate the similarity between current problem and recorded previous problems (see Figure 4). We employ traditional *tf-idf* and *Cosine* [V79] measures from information retrieval to calculate similarity values. In the database, all the symptoms of a root cause are combined as a mixed symptom $S_i = S_{i,1} + S_{i,2} + \cdots + S_{i,n}$. The current symptom $S_{current}$ or each of the recorded symptoms S_i is represented as a vector of term frequency. The basic assumption here is that if a state is a good candidate to the current problem, it is highly likely that it caused some problems with similar symptoms in the past. The calculated similarity values are used to rank the candidate registry entries.

Intersection-Ranking, Diff-Ranking & Trace-Ranking

We collected 74 registry-related real-world problems reported by our colleagues and users of web support forums. These problems are independent from the PC-Genomics Database we are building. For each problem, we recorded its symptom description, trace set, and differencing set. We also calculated the intersection set and root cause rankings with the Strider tool. There are three points to apply our ranking schemes. First, we can apply ranking on the intersection-set and it is called *intersection-ranking*. This ranking is expected to produce better result since the size of intersection is relatively small and the ranking cost is also small. Second, when no trace data is available, we can directly apply *diff-ranking* to the diffing set. Finally, when no diffing data is available, we can also directly rank the trace set, which is called *trace-ranking*.

Relaxed Root Cause Matching in the Database

If both the value name and path name of a root cause can be matched in the PC-Genomics database, we call it *value-matched*. If only the path name is matched, we call it path-matched. Otherwise, we call it not-matched. For example, the configuration state with path name "hkey_classes_root\.jpg," and value name "(Default)" is considered value-matched, if "hkey_classes_root\.jpg\(Default)" can be found in the database. Or else if "hkey_classes_root\.jpg" is in the database, it is called path-matched. If none of them are in the data base, this configure is considered to be not-matched. To our experience, this relaxed matching criterion can increase the problem coverage of our method and do little harm to its accuracy.

Because the "registry dictionary" covers only 66 of the 74 root causes, the remaining eight root causes could not be recognized in the knowledge source free text. The PC Genomic database extracted from PSS log covers the root causes of 59 problems, while the database from KB covers 37.

Ranking Result

The result of a diagnosis processes is actually a rank of candidate root causes, ordered by their likelihood of being the actual one. Obviously, the real root cause should be ranked as high as possible in order for this approach to be effective. Usually, a ranking less than 5 is preferred. For each method, we sorted the cases by the ranking of their actual root causes. With these ranking curves, we can easily compare the diagnostic effectiveness of different methods.

Figure 5 contains the ranking curves using PSS log. We can see that our method efficiently increases the diagnosis accuracy with intersection data. Only nine out of 59 real root-causes rank more than 5. However, ranking with only differencing data or trace data is not very accurate.

Figure 6 contains the ranking curves using KB. The ranking curves show that our knowledge database from KB articles still gives good accuracy with intersection data and differencing data, but the ranking with trace data deteriorates a lot.

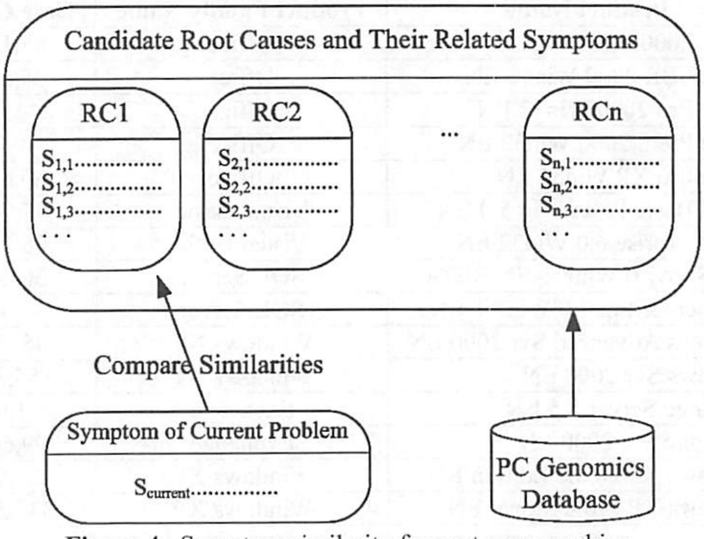


Figure 4: Symptom similarity for root cause ranking.

Discussion

In this section, we discuss the strengths and weaknesses of the method.

One-to-many Mappings Between Symptoms and States

Problems with the same or similar symptom(s) may be caused by different registry entries. For instance, we manually checked the PSS log and found that 17 entries have been reported to cause the "Cannot open Word document" problem (see Table 3). If we use only the symptom-based methods for troubleshooting, we will get multiple possible root causes for a problem. In our approach, state information, like a filter that is orthogonal to the symptom description, can point out the root cause efficiently.

Problem Coverage

The effectiveness of our approach depends on the problem coverage of our database. We need two things to achieve this goal: building a "registry dictionary" with good problem coverage, and extracting information from a knowledge source of good problem coverage.

In PSS log data, only about 0.5% registry entries are ever reported to cause problems (i.e., 4,837 of 898,546). In KB data, only about 0.1% entries are ever reported (i.e., 996 of 898,546). If we consider these entries as a filter, they would be very efficient in scaling down the candidate root cause set.

Among the 4,837 fragile registry entries from the PSS log, only a few entries cause problems frequently. Most entries have small numbers of occurrences (see Figure 7). They approximately follow a Zipf distribution [Z49]. Even if the PC-Genomics Database contains only a portion of the known problems, it can still greatly reduce the real-world enterprise support cost because the most costly problems are well covered.

Since the state-symptom database is flexible, the coverage problem can be further alleviated. If we find

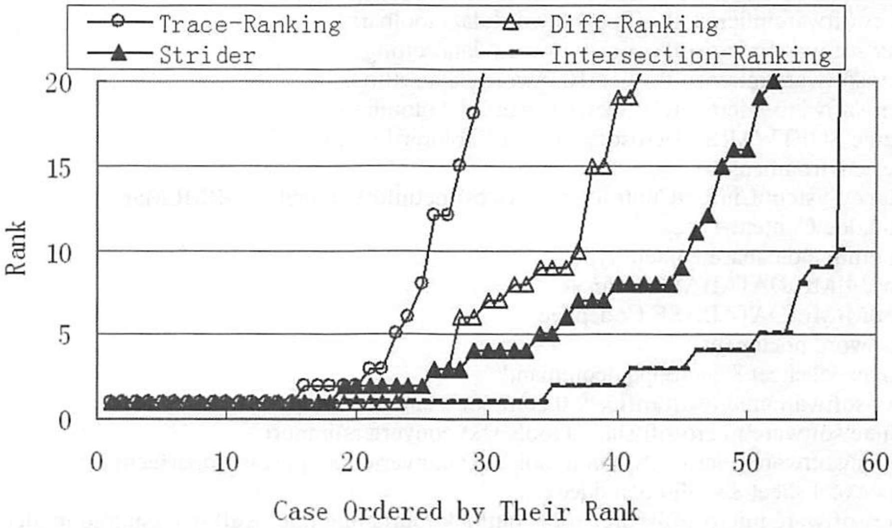


Figure 5: PSS log ranking results.

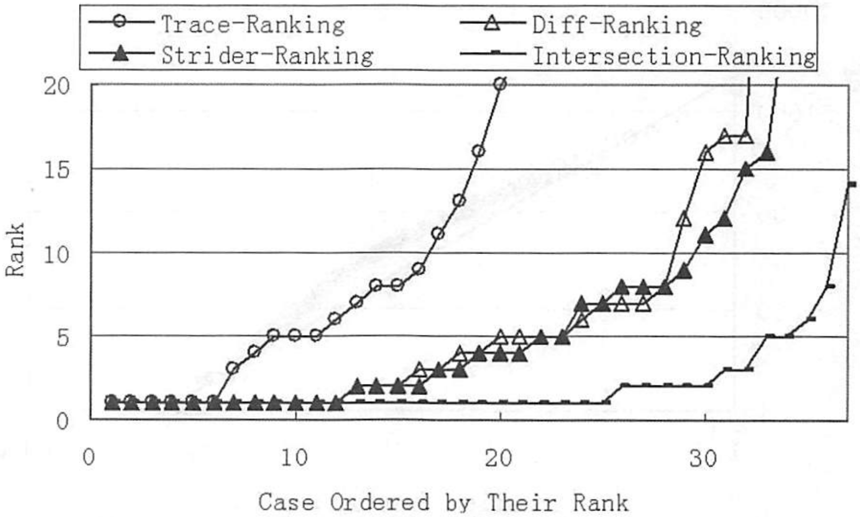


Figure 6: KB ranking results.

a common problem outside the dictionary or knowledge source, we can simply add its symptom and state into the database manually. But as long as we can only handle problems for which the root-cause entries have been found before, Strider needs to be used to find those root causes for the first time.

The Percentage of Registry-Related Problems

Directly calculating the percentage of registry-related problems from the number of cases which have cited a registry key yields a very small number: 6.2% (i.e., 143,157 of 2,311,492). One reason behind this is that the engineers often cite a registry related KB article and ask the user to do what the article says without specifying the registry key name. Another big reason is that finding root causes of Registry problems were extremely hard before Strider.

With the help of KB information for each case, we can get a better estimation. When a case is closed, the engineer is required to cite a KB article ID as the description of the case. About 8.8% of KB articles (i.e., 12,464 of 142,448) contain the keyword "registry." These "registry" KB articles cover 27% of the cases that ever cite KB. We manually verified 40 of these KB articles, and found that 15 of them are not actually registry problems. They may be just providing registry-related knowledge or using registry as a problem solving method. Thus, the overall registry problem percentage is approximately 17% (i.e., 27%*25/40).

Summary

We have proposed a novel solution to combine the traditional symptom-based troubleshooting method and relatively new state-based troubleshooting method. It adds some overhead of data collection to the user, but it can solve some previously hard-to-solve problems. So we prefer to treat the PC-

hkey_users\.default\Software\Microsoft\Office\8.0\Outlook\Options\Mail
hkey_current_user\software\microsoft\office\9.0\word\data\toolbars
hkey_current_user\software\microsoft\office\9.0\word\data\settings
hkey_current_user\software\microsoft\office\10.0\word\data\settings
hkey_current_user\software\microsoft\office\10.0\word\data\toolbars
hkey_local_machine\SOFTWARE\Microsoft\Internet Explorer\Plugin
hkey_current_user\environment
hkey_local_machine\System\CurrentControlSet\Services\Inetinfo\Parameters\MIMEMap
hkey_classes_root\.doc\Content Type
hkey_classes_root\mime\database\content type
hkey_classes_root\MIME\DATABASE\Charset
hkey_classes_root\MIME\DATABASE\Codepage
hkey_classes_root\word.document
hkey_classes_root\excel.sheet.8\shell\open\command
hkey_current_user\software\microsoft\office\9.0\common\general\startup
hkey_local_machine\software\microsoft\shared tools\text converters\import
hkey_local_machine\software\microsoft\shared tools\text converters\import\wordperfect6\&x
hkey_classes_root\excel.sheet.8\shell\open\ddeexec
hkey_current_user\software\microsoft\shared tools\outlook\journaling\microsoft word\autojournaled

Table 3: Registry key root causes of "Cannot open Word document" problem.

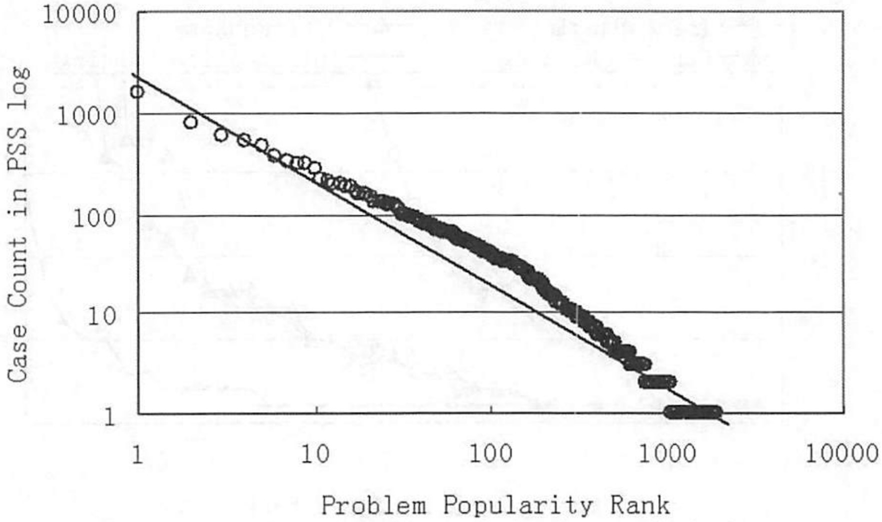


Figure 7: Occurrence of 1,913 value-matched registry entries in the PSS Log.

Genomics technique as a backup for the regular methods and to use it only when regular methods fail. Our future work will exploit other types of system information which can give the troubleshooting process better problem coverage and make it more automatic.

Acknowledgement

would like to express our sincere thanks to our shepherd Æleen Frisch for her valuable feedback, to Aaron Johnson, Chad Verbowski, and Archana Ganapathi for their analysis of the test cases, and to the many colleagues who helped collect the registry snapshots from 50 computers.

Author Information

Ni Lao is currently a graduate student in School of Software at Tsinghua University in China. He received his B.S. in Electronic Engineering from Tsinghua University in 2003. His current research focuses on automated system management using data mining and pattern recognition methods. He can be reached at noon99@mails.tsinghua.edu.cn.

Ji-Rong Wen is a researcher in Microsoft Research Asia. He received his Ph.D. in Computer Science from the Institute of Computing Technology, the Chinese Academy of Science in 1999. He joined Microsoft in July 1999. His current research interests are data management, information retrieval (especially Web search), data mining and system management.

Wei-Ying Ma received the B.S. degree in electrical engineering from the national Tsing Hua University in Taiwan in 1990, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of California at Santa Barbara in 1994 and 1997, respectively. He joined Microsoft Research Asia in April 2001 as the Research Manager of the Information Management and Systems Group. Prior to joining Microsoft, he was with Hewlett-Packard Laboratories at Palo Alto. From 1994 to 1997 he was engaged in the Alexandria Digital Library (ADL) project in University of California at Santa Barbara while completing his Ph.D. Dr. Wei-Ying Ma serves as an Editor for the ACM Multimedia System Journal and Associate Editor for the Journal of Multimedia Tools and Applications published by Kluwer Academic Publishers. His research interests include Internet search, information retrieval, content-based image retrieval, intelligent information systems, adaptive content delivery, and media distribution and services networks.

Yi-Min Wang is the manager of the Systems Management Research Group at Microsoft Research, Redmond. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include systems and security management, fault tolerance, home networking, and distributed systems.

References

[Ande95] Anderson, E. and D. Patterson, "A Retrospective on Twelve Years of LISA Proceedings," *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII),* USENIX, p. 95, 1999.

[Apple] *Knowledge Base*, http://kbase.info.apple.com .

[BugNet] BugNet, *BugNet*, http://www.bugnet.com .

[CKF02] Chen, M., E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic, Internet Services," *Proc. Int. Conf. on Dependable Systems and Networks (IPDS Track)*, 2002.

[Gan04] Ganapathi, A., Yi-Min Wang, Ni Lao, and Ji-Rong Wen, "Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems," to appear in *Proc. IEEE DSN/DCC*, June 2004.

[Gray03] Gray, J., "What Next? A Dozen Information-Technology Research Goals," *Journal of the ACM*, Vol. 50, Num. 1, pp. 41-57, January 2003.

[LC01] Larsson, M. and I. Crnkovic, "Configuration Management for Component-based Systems," *Proc. Int. Conf. on Software Engineering (ICSE)*, May 2001.

[MSKB] Microsoft Corporation, *Microsoft Knowledge Base*, http://support.microsoft.com .

[Qie03] Qie, X.-H., Sanjai Narain, "Using Service Grammar to Diagnose BGP Configuration Errors," *Proc. Usenix Large Installation Systems Administration (LISA) Conference*, pp. 237-246, October 2003.

[Redhat] Redhat Corporation, *Redhat Support Forums*, http://www.redhat.com/support/knowledgebase/forums/ .

[RSB03] Redstone, J. A., M. M. Swift, B. N. Bershad, "Using Computers to Diagnose Computer Problems," *Proc. HotOS*, 2003.

[SR] Windows XP System Restore, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/windowsxpsystemrestore.asp .

[SR00] Solomon, D. A. and M. Russinovich, *Inside Microsoft Windows 2000*, Microsoft Press, Third Edition, Sept 2000.

[V79] van Rijsbergen, C. J., *Information retrieval*, Butterworths, Second Edition, London, 1979.

[W03] Wang, Y.-M., Verbowski, Chad., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., and Zhang, Z., "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. Usenix Large Installation Systems Administration (LISA) Conference*, pp. 159-171, October 2003.

[Z49] Zipf, G. K., *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology*," Addison Wesley, Cambridge, MA, 1949.

LifeBoat – An Autonomic Backup and Restore Solution

Ted Bonkenburg, Dejan Diklic, Benjamin Reed, Mark Smith – IBM Almaden Research Center
Michael Vanover – IBM PCD
Steve Welch, Roger Williams – IBM Almaden Research Center

ABSTRACT

We present an innovative backup and restore solution, called LifeBoat, for Windows machines. Our solution provides for local and remote backups and "bare metal" restores. Classic backup systems do a file system backup and require the machine to be installed before the system can be restored or they do a block for block backup of the system image which allows for a "bare metal" restore, but makes it hard to access individual files in the backup. Our solution does a file system backup while still allowing the system to be completely restored onto a new hard drive.

Windows presents some particularly difficult problems during both backup and restore. We describe the information we store during backup to enable the "bare metal" restore. We also describe some of the problems we ran into and how we overcame them. Not only do we provide a way to restore a machine, but we also describe the rescue environment which allows machine diagnostics and recovery of files that were not backed up. This paper presents an autonomic workgroup backup solution called LifeBoat that increases the "Built-In Value" of the PC without adding hardware, administrative cost, or complexity. LifeBoat applies autonomic principles to the age old problem of data backup and recovery.

Introduction

Supporting PC clients currently represents roughly 50% of overall IT cost (IGS 2001). This number is larger than both server (30%) and network related costs (20%). This provides the motivation for an autonomic approach to reducing the cost of PC clients. So far, thin clients have repeatedly failed in the marketplace. IT attempts to "lock down" PC clients have not been accepted. In addition, attempts to control the client from the server have failed due to the fact that clients sometimes get disconnected. Fat clients, however, continue to prosper and increase in complexity which drives the maintenance cost up. We believe autonomic clients are critical components of an overall autonomic computing infrastructure. They will help lower the overall cost of ownership and reduce the client down time for corporations.

The secure autonomic workgroup backup and recovery system, LifeBoat, provides data recovery and reliability to a workgroup while reducing administrative costs for Windows 2000/XP machines. LifeBoat provides a comprehensive backup solution including backing up data across the peer workstations of a workgroup, centralized server backup, and local backup for disconnected operation. In addition, it provides a complete rescue and recovery environment which allows end users to easily and conveniently restore downed machines. The LifeBoat project increases the Built-in Value of the PC without adding hardware, administrative, or complexity costs. By leveraging several autonomic technologies, the LifeBoat project increases utility while reducing administrative cost.

In this paper we first describe the backup portion of LifeBoat. This is split into two sections, the first of which focuses on network backup. LifeBoat leverages a research technology called StorageNet to seamlessly spread backup data across the workstations of a workgroup in a peer-to-peer fashion. We then create a scalable road map from workgroup peer-to-peer to a centrally managed IT solution. The second backup section focuses on backing up to locally attached devices which is a requirement for disconnected operation. We then describe the complete rescue and recovery process which simplifies recovery of files and directories as well as providing disaster recovery from total disk failure. Next we describe a centralized management approach for LifeBoat and how LifeBoat can fit within a corporate environment. We conclude with performance measurements of some example backup and restore operations.

Backup

LifeBoat supports a number of backup targets such as network peers, a dedicated network server, and locally attached storage devices. The Autonomic Backup Program is responsible for creating a backup copy of a user's file system in such a way as to be able to completely restore the system to its original operating state. This means that the backup must include file data as well as file metadata such as file times, ACL information, ownership, and attributes. Our backups

are performed file-wise to enable users to restore or recover individual files without requiring the restoration of the entire machine such as with a block based solution. Finally, the backups are compressed on the fly in order to save space.

The Autonomic Backup Program performs a backup by doing a depth first traversal of the user's file system. As it comes to each file or directory, it creates a corresponding file in the backup and saves metadata information for the file in a separate file called attributes.ntfs which maintains the attributes for all files backed up.

There is special processing required for open files locked by the OS. The backup client employs a kernel driver to obtain file handles for reading these locked files. This driver stays resident only for the duration of the backup. When a backup is completed, the client generates a metadata file to describe the file systems which have been backed up. This "usage" file contains partition, file system, drive lettering, and disk space information. The existence of the usage file indicates that the backup was successful.

The output of the backup and format of the backup data depends on the target. For a network backup, the data is stored using a distributed file system known as StorageNet. StorageNet has some unique features which make it especially suited for our peer-to-peer and client-server backup solutions. For a backup to locally attached storage, the backup is stored in a Zip64 archive.

StorageNet Overview

The storage building block of our distributed file system, StorageNet, is an object storage device called SCARED [2] that organizes local storage into a flat namespace of objects identified by a 128-bit object id. A workstation becomes an object storage device when it runs the daemon to share some of its local storage with its peers. While the object disks we describe here are similar to other object based storage devices [2, 3, 4, 8], our model has much richer semantics to allow it to run in a peer-to-peer environment.

Clients request the creation of objects on SCARED devices. When an object is created, the device chooses an object id to identify the newly created object, marks the object as owned by the peer requesting creation, allocates space for it, and returns the object id to the client. Clients then use the object id as a handle to request operations to query, modify, and delete the object.

An object consists of data, an access control list (ACL), and an info block. ACLs are enforced by the server so that only authorized clients access the objects. The info block is a variable sized attribute associated with each object that is atomically updated and read and written by the client. The info block is not interpreted by the storage device.

One special kind of object creation useful in backup applications is the linked creation of an object. We implement hard links by passing the object id of an existing object when requesting creation. A hard link shares the data and ACL of the linked object, but has its own info block. These linked objects allow us to not only create hard links to files, but also to directories. Hard linked objects are not deleted until the last hard link to the object is deleted.

There are two kinds of objects stored on SCARED devices. File objects have semantics similar to local files. They are a stream of bytes that can be read, written, and truncated.

Directory objects are the other kind of object; they are an array of variable sized entries. Entries are identified by a unique 128-bit number, the etag, set by the daemon as well as a unique 128-bit number, the ltag, set by the client. The client chooses an ltag by hashing the name of the file or directory represented by an entry. The entry also has variable sized data associated with it that can be read and set atomically.

Later we will describe how these objects are used to build a distributed file system, but here we need to point out that the storage devices only manage the allocation and access to the objects they store. They do not interpret the data in those objects, and thus, do not know the relationships between objects or know how the objects are positioned in the file system hierarchy. Because the data stored on the storage devices is not interpreted, the data can be encrypted at the client and stored encrypted on the storage devices.

SCARED devices also track the allocations of objects for a given peer to enforce quotas. Later we will explain why quota support is needed, but for now it is important to note this requirement on the storage devices.

Along with object management, storage devices also authenticate clients that access them. All communication is done using a protocol that provides mutual authentication and allows identification of the client and enforcement of quotas and access control. Note that communication only occurs between the client and storage device; storage devices are never required to communicate with each other.

Clients use data stored on the object storage devices to create a distributed file system. The clients use meta-data attached to each object and directory objects to construct the file system. The directory entries are used to construct the file system hierarchy and the info blocks are used to verify integrity.

Figure 1 shows the layout of the directory entries as interpreted by the client. The first three fields are maintained by the storage device. The other fields are stored in the entry data and thus stored opaquely by the storage device. The client needs to store the filename in the entry data since the ltag is the hash of the filename, which is useful for directory lookups, but

the actual filename is needed when doing directory listings. The other important piece of information is the location of the object represented by the entry or, if the entry is a symbolic link, the string representing the symbolic link which is stored in the entry data.

<table>
<tr><td rowspan="2">Entry tag</td><td rowspan="2">Lookup tag</td><td rowspan="2">Version</td><td rowspan="2">Filename</td><td rowspan="2">Location type</td><td>OID</td><td>Hostname</td></tr>
<tr><td colspan="2">Symlink</td></tr>
</table>

Figure 1: Layout of the directory entry.

Figure 2 shows a fragment of the distributed file system constructed using the structures outlined above. The first device contains two objects. The first object is a directory with two entries. The first entry represents a directory stored on the second device. The second directory entry is a file that is stored on the same device.

Peer-to-Peer Network Backup

In the peer-to-peer case, our system backs up workstation data onto other workstations in the workgroup. This is accomplished by defining a hidden partition on each workstation that can be used as a target of the backup. The architecture of the software components in the system is completely symmetric. Each workstation runs a copy of the client and the StorageNet server. In this way each station serves as both a backup source and target. In addition, each station runs a copy of the Lifeboat agent process. This always runs, provides, and serves the web user interface that constitutes the policy tool to allow the user to make changes to backup targets, select files for backup, and set scheduling times. At the appointed time, this process will invoke the backup client program as well. The hidden partition is created during the installation process and is completely managed by the StorageNet server on each station. The customer uses the client software to specify what data to backup and on what schedule. The target of the backup is determined by the system and can be changed by the customer on request.

In the case of an incremental backup, our StorageNet distributed file system offers some very strong advantages over traditional network file systems. For example, one feature which we use a great deal is the ability to create directory hard links. In this way, if an entire subtree of the file system remains unchanged between a base and an incremental backup, we can simply hard link the entire subtree to the corresponding subtree in the base backup. When individual files remain unchanged, but their siblings do not, we can hard link to the individual files, and create new backup files in the directory. This unique directory and file hard linking ability allows each backup in our file system, both base backup and incremental backups, to look like an entire mirror image of the file system on the machine being backed up. Each incremental backup only takes up the same amount of space as what has changed between backups.

In the local case, incremental backups look like a subset of the file system. Pieces of the file system that did not change are simply not copied into the zip. In order to distinguish between files that are unchanged and files that have been deleted, we keep a list of files which have been deleted in "DeletedFiles.log." This is used during the restore to know which files not to copy out of the base.

For example, consider backing up the file helloworld.txt. In the remote scenario, this file is copied to our StorageNet distributed file system. The filename, file data, file times, and file size are all set in the StorageNet file system.

File dates and sizes are not stored redundantly in this case because the cost of looking them up later during an incremental is free. This is because during a remote incremental, we are also doing a depth first traversal on the base backup. File ACL, attributes, and ownership information is placed into attributes.ntfs file for use during restore. The short- name data is stored in the directory entry for this file. Although StorageNet has no 8.3 limitations, it makes provision for this information to maintain full compatibility with Windows file systems.

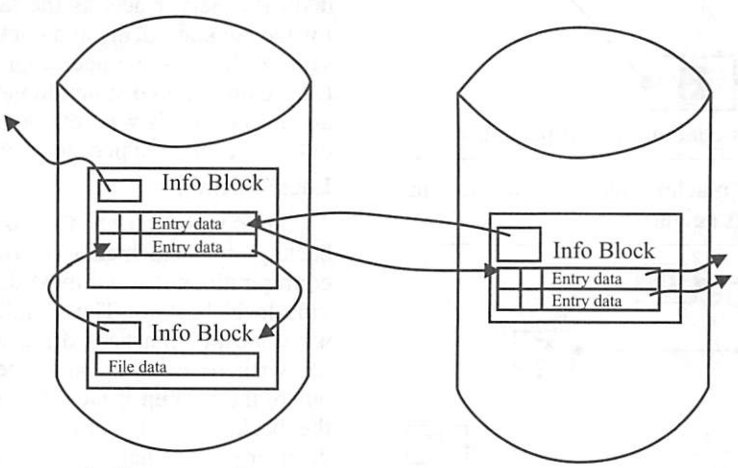


Figure 2: An example file system fragment stored on directory objects on two storage devices.

Why We Need Quotas

Since all the peers store their data remotely, if any peer fails it can recover from its remote backup. It is tempting to randomly spread a given peers backup across all of its peers. Spreading this way gives us some parallelism when doing backups and should speed our backup. However, if we do spread a given machine's backup uniformly across its peers, we cannot tolerate two failures since the second failure will certainly lose backup data that cannot be recovered.

Instead of spreading the data across peer machines, we try to minimize the number of peers used by a given machine for a backup. Thus, if all machines have the same size disks, when a second failure happens there will only be a $1 over (n˜-˜1)$ chance that backup data is lost for a given machine.

Unfortunately, we cannot assume that all peers have the same sized disks. Thus some peers may store the backup data of multiple clients, and other peers may use multiple peers to store their backup. If the disk sizes are such that a peer's backup must be stored on multiple peers and those peers in turn store backups from multiple peers, the backups can easily degenerate into a uniform backup across all peers unless some form of quotas are used.

Peer Backup Scenarios

The number of scenarios that are supported by this solution is virtually innumerable. However, there are some attributes that constitute simple scenarios. For example, we can consider the most simple scenario in the peer-to-peer case to be the completely symmetric homogeneous case where all stations provide a hidden partition that is equal in size to their own data partition, and each stations data is backed up to a neighboring station. Figure 3 shows an example for three workstations.

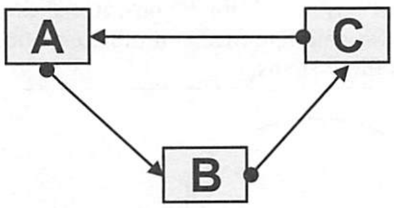


Figure 3: Three workstation peer-to-peer case.

In this case every machine backs up its data in the hidden partition of its neighbor.

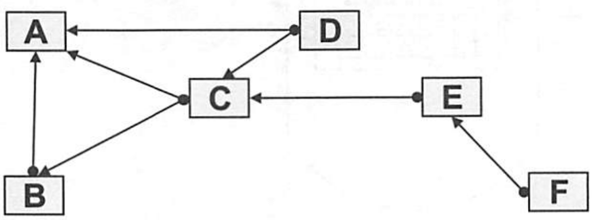


Figure 4: Non-homogenous/non-symmetric example.

Figure 4 shows a more complicated scenario. In this case, the following statements hold true for the backup group:

- B holds all of A's data and portions of the data of C
- A holds all of B's data and portions of the data of C and D
- C holds all of E's data and portions of the data of D
- D could be a laptop and stores parts of its data on A and C
- E holds all of F's data
- F (as well as possibly other stations) has available target space for a new entrant in the group

Either of these scenarios could have resulted from:

- autonomic system decisions based on the sizes and allocations of a heterogeneous group of workstations
- user-selection specifies the target of the backup

Obviously these two scenarios are not exhaustive. Configurations of arbitrary complexity are supported. We intend to develop heuristics and user interface methods to reduce possible complexity and allow the customer to efficiently manage the backup configuration.

Dedicated Server Network Backup

One of the big advantages of using dedicated servers as opposed to peers is the availability of service. Because peers are general purpose user machines, they may be turned off, rebooted, or disconnected with a higher probability than with dedicated servers. In a large enterprise environment using a dedicated server approach can guarantee backup availability. Machine stability is important when trying to do backups. Dedicated servers are also easier to manage because of their fixed function. Machines are also easier to update and modify by an admin staff if they belong to the IT department rather than users.

The dedicated server solution uses StorageNet in a similar fashion as the peer-to-peer approach. The dedicated server acts as the target StorageNet device for the backup clients and backup data is stored in the same fashion as the peer approach. Indeed, the architecture makes no distinction between dedicated servers and peers. In this way, the dedicated server solution is only a special case peer-to-peer usage scenario.

Local Backup

For mobile users the ability to perform regular backups to local media is critical. There are several configurations that we must deal with in order to provide local backup. The simplest one is for a system with one internal hard drive which contains the data we wish to back up and one additional hard drive where the backup is stored. The hard drive containing the backup can be either an external USB/Firewire drive or internal hard drive. The user is also allowed to perform backup locally to the source hard drive. In

this case we use a file system filter driver to protect the backup files. While this form of backup wont protect the user from hard drive failures, it will allow recovery from viral attacks or software error.

The format of the backup is rather simple. We use a simple directory structure. The main directory is called LifeBoat_Local and for every machine backed up to the drive we add another sub directory. This sub directory, for example test, will contain multiple directories and files. The most important file contains the UUID of the machine that is backed up and is called the machine file. It contains the serial number of the machine and UUID as returned by DMI [1]. We use this file during the restore procedure to automatically detect backups. A sample machine file is in Figure 5.

```
2658N5U AKVAA2W
00F7D68B-0AA0-D611-88F2-EDDCAE30B833
```

Figure 5: Typical machine file.

The first time a local backup is run, we create a directory called base and place it in LifeBoat_Localest. Additional backups are placed in directories called Incremental 1, Incremental 2, etc. The number of incremental backups is user configurable, with the default value set at five. The full directory structure can be seen in Figure 6. Each of the directories such as base and Incremental 1 contain the following files: usage, attributes.ntfs, backup.lst, and some zip files. In the case of a filesystem that is less than 4 GB compressed, a single zip file, backup.zip suffices. Otherwise, the Zip64 spanning standard is used.

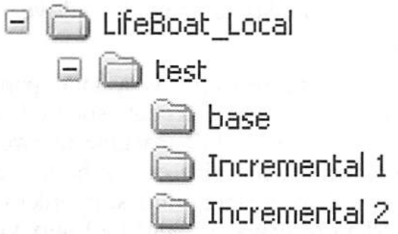


Figure 6: Typical directory structure.

The first line of the usage file lists descriptions of columns inside the usage file: drive letter, file system type, size of the partition, amount of used space and amount of backed up data. In the next line is an OS descriptor which is important for post processing after restore. Possible descriptors are WinXp, Win200, WinNT4.0, WinNT3.5, Win98, Win95 and WinME. Lines that follow give information about each partition in the system. They are used during restore process.

The attributes.ntfs file is of importance only when backing up/restoring NTFS partitions and is not used if the partitions are not NTFS. The attributes.ntfs file contains all file attributes as well as ACL, SACL, OSID and GSID data. We write the data during backup and restore it during the restore post processing step. Backup.zip contains the actual backup of all files. Through extending ZIP functionality to use the current Zip64 specification, we are able to create ZIP files that are very large, dwarfing the original 2 GB limit. If the backup is greater then 4 GB zipped we can create multiple backup files (backup.zip, backup.001, etc.) using the Zip64 spanning standard. We chose 4 GB as our spanning limit in order to allow these files to be read by FAT32 files systems.

For example, let's imagine we are doing a base backup and come to the file "helloworld.txt" which contains data as well as some ACL information. This file would be added to the backup.zip file and compressed, taking care of the filename, file data, modification date, and file size. The file dates and size are also placed in a metadata file, backup.lst, to be used later when creating incremental backups to determine whether the file has changed and needs to be backed up again. File ACL, attributes, and ownership information is placed into the attributes.ntfs file for use during restore. Finally, the short name for this file, for example "hellow~1.txt", is stored in the comments section of the zip file. Preserving shortnames across backup and restore turns out to be very important even in later Windows versions. Some Windows applications still expect the short names for files to not change unless the long filename changes as well.

A special case of local backup is the backup to yourself case. In this case we have only one hard drive and we want to backup the data to the same drive we are backing up. In the simple case we have multiple partitions on the hard drive, for example we backup drive C to drive D. In a more complex case where we have to deal with a single partition we backup C to C. As far as backup is concerned this is not problematic, however during restore we have to deal with some very specific problems related to NTFS partitions and the lack of write support under Linux.

Rescue and Recovery

A significant portion of the LifeBoat project focuses on client rescue and recovery. This includes several UI features for Windows as well as a bootable Linux image. The rescue operations allow a user to perform diagnostics and attempt to repair problems. Recovery enables the user to restore individual files or even perform a full restore in the case of massive disk failure.

Single File Restore

When the system is bootable, it is possible to restore a single file or a group of files from within Windows [6]. In keeping with the autonomic goal of the system, the user interfaces for this system are minimal. From Windows, the restore process uses a simple browser interface to StorageNet using the browser protocol istp://. A screenshot of the istp protocol is below in Figure 7. We have also written a namespace extension for StorageNet which behaves like the ftp namespace extension which ships with Windows. An example

screen looks almost identical to that for ftp:// and uses the analogous Copy-Paste commands (see Figure 8).

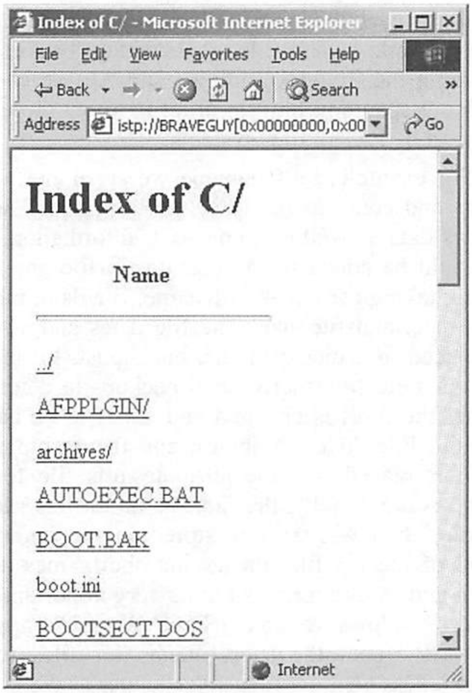


Figure 7: The istp:// protocol.

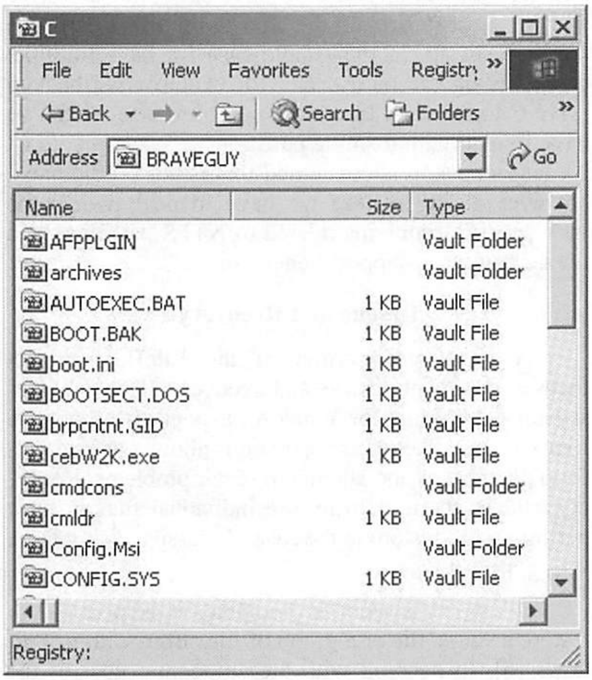


Figure 8: StorageNet namespace extension.

Rescue

The LifeBoat Linux boot CD provides various software services that can be used for systems maintenance, rescue, and recovery. The distribution works in almost any PC and can be booted from a number of devices such as a CD-ROM drive, USB keyfob, local hard drive, or even over the network. The CD includes over 101 MB of software including a 2.4.22 kernel, Xfree86 4.1, full network services for both PCI and PCMCIA cards and wireless connectivity.

An important part in the design of the bootable Linux CD was rescue functionality. We wanted to provide the user with at least a rudimentary set of functions which would enable him to diagnose, report, and fix the problem if at all possible. As part of the CD we included the following set of rescue functions: PCDoctor based diagnostics which lets us run an all encompassing array of hardware tests, AIM as way of quickly communicating with help available online, and the Mozilla web browser. We also developed an application which finds all bookmarks on the local drive, in the local backup, and in the remote backup and makes them available for use in Mozilla. This provides the user with the list of bookmarks that he is used to. At the same time we add a selection of bookmarks which can be custom tailored for a specific company to include their own links to local help desk sites and other useful resources.

Even in the face of disaster, an important issue to keep in mind is that a damaged hard disk may still contain some usable data. In the case of a viral attack, boot sectors and system files could be compromised but the user data could be left intact. Performing a full system restore would overwrite any changes made since the last backup. For this case we created an application that browses through all documents that were recently accessed and allows the user to copy them to a safe medium such as a USB keyfob or hard drive.

Recovery

Full machine recovery is a vital part of any backup solution. The Lifeboat solution uses its bootable Linux CD for full machine restore. This is necessary when the machine cannot be booted to run the Windows based restore utilities. In order to use the CD for system recovery we added a Linux virtual file system (VFS) implementation for StorageNet. Located on the CD is our Rapid Restore Ultra application which is used to restore both local and remote backups. Rapid Restore Ultra is written in C and uses QT for the UI elements [9]. The application comes in two flavors. The first one is intended for a novice user that has no deep knowledge of systems management issues and just wishes to restore the data. The second version is intended for knowledgeable system administrators or advanced users that have deep knowledge of internal systems functioning. The novice user just restores the latest backup and the application determines how the backup is to be restored. Advanced users can select any backup on the discoverable network or local devices, as well as forcing the discovery of backups on a non local networks by entering the IP address or name of a potential server. The user can then manually repartition the drives, and assign drive letters and data.

Drive partitions can even be set to different sizes than were originally backed up. This way the user has full control of the restore process.

Performing a Full Restore

The first step towards machine recovery is creation of new partitions. To do that we can either use the description of partitions from the usage file in the last backup or let the user decide the partition sizes and types. The usage file is used for both local and remote restores and gives all the information about the old partition table. In order to determine new partition sizes we use a simple algorithm that takes into account old partition size, new partition size, the number of partitions, and the percentage of usage.

Before we write the partition table we have to make sure we have a valid Master Boot Record (MBR). To be sure we dump our MBR onto the first 32 sectors of the drive. It is important to keep in mind that the MBR that is written at this moment has no partition information. If there were any partition info at this step in the MBR, we couldn't be sure that the disk geometry we are using is correct.

After writing the MBR, we write the partition table. After the partition table is successfully written we have to format all partitions. One of the issues is the need to support all of the current Windows file systems such as FAT, FAT 16, FAT 32 and if possible, NTFS. Linux can format all of the FAT file systems, but can't create bootable FAT file systems. In order for a file system to be able to boot, the master boot record must point to a valid boot sector. Support for NT, WIN2k and XP is provided through the use of our application. We pieced together information about Windows boot sectors and after long debugging found a way to create valid boot sectors on our own. The reason why we are unable to use the original boot sectors from a previously backed up machine is simple. Boot sectors are dependent on partition sizes and geometry, thereby requiring us to create them every time we repartition. Another reason for not restoring the boot sector from a backup is that boot sectors are a favorite hiding place for viruses.

After the disk is formatted and the boot sectors are written, we start the client application to restore the data. If we are performing remote restore, the client connects to the server and upon successful authorization the files, including the operating system, are copied to the local partition. This process is repeated for as many partitions as necessary. After all the files are transferred the machine is rebooted and available for work. Here is a summary of the steps performed in this process:

- Write general MBR
- Write new partition table
- Format partitions
- Mount boot partition
- Start Sys16 (for FAT16) or Sys32 (for FAT32) to create valid boot sector
- Transfer system files
- Copy remaining files
- Unmount partition

If we are performing a local restore there are multiple issues we have to face. The first problem is related to having the backup located on the same drive we are trying to restore to. If this is the case, we are unable to reformat the partitions and also we cant change partition sizes. Another issue is related to NTFS support in Linux. Lets say we are backing up to the C drive and it is formatted NTFS. When the restore starts it will find the backup on the first partition and notice that the partition type is NTFS. While Linux has very good support for reading NTFS file systems it has minimal support for writing NTFS. The solution to this, which is detailed in the next section, is a technique for formatting an existing NTFS partition as FAT32 while preserving the backup files.

Once all the preparatory steps are successfully completed, we start unzipping data to the desired partition. If we have only a base backup, the restore process ends when unzipping of the base backup.zip file is completed. In case of incremental backups the restore process is more complicated. Suppose we have three incremental backups and the base backup. If we wish to restore the third incremental backup, we start by unzipping the backup.zip located in the Incremental 3 directory. Then we unzip the backup.zip located in the Incremental 2 directory and so on. We do this until we have finished the backup.zip in the base directory. Each time we have to make sure that no files get overwritten.

Once unzip finishes we have to create post processing scripts that will run immediately following Windows boot. We have to take care of two problems: proper assignment of drive letters and NTFS conversion. In case of a backup with more than three partitions we cant be sure that once Windows comes up it will assign correct drive letters to their respective partitions. It is also possible that we didnt use C, D, or E as drive letter in Windows but for example C, G, and V. While performing backup we add a file called driveletter.sys to each drive on the hard disk. This file only contains the drive letter. The first thing after restore we need to do when windows comes up is change drive letter names. This is done easily by changing registry entries to values we read from driveletter.sys and doesnt even require a reboot. A second problem is related to NTFS partitions. When we restore we create our partitions to be FAT32 and format them accordingly. Once restore is completed and drive letter assignment has run its course, we have to convert those partitions back to NTFS. This is accomplished using the convert.exe utility that is supplied in Windows.

Upon completing conversion of the drives to NTFS we have to set attributes and ACLs for all files on that drive. We wrote a simple application that reads

the content of the attributes.ntfs file and sets ACL, System ACL, Owner SID, and Group SID as well as file creation/modification times. This application lets us set all file attributes. Upon completion it deletes the attributes.ntfs file and exits. That is also the last step in post processing.

Same Partition NTFS Backup

In order to overcome the lack of write support in the Linux NTFS driver we developed a technique whereby an NTFS partition can be formatted as FAT32 while preserving the backup files. This is in essence converting an NTFS partition to FAT32.

The conversion process consists of a number of steps. First a meta file which contains data about the file to be preserved is created. Next the set of parameters for formatting the partition as FAT32 is carefully determined. The next step is running through all of the files to be preserved and relocating on disk only those portions that need to be moved in order to survive the format. The partition is formatted and the files are resurrected in the newly created FAT32 partition. Finally, directories are recreated and the files are renamed and moved to their original paths.

The set of files that need to be preserved must be known a priori. In the case of the LifeBoat project, this consists of a directory and a small set of potentially large files. The first step is to create the meta file which contains enough information to do a format while preserving these files. The meta file may be created immediately after a backup from within Windows or, if the NTFS partition is readable, it is created in a RAM disk from within the Linux restore environment.

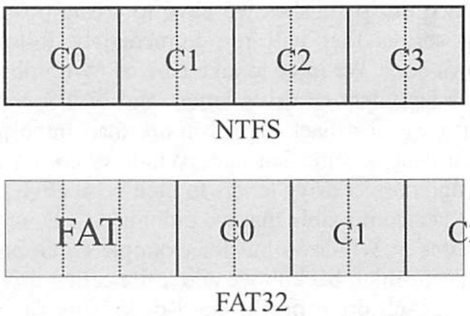


Figure 9: The top partition shows the first four clusters of an NTFS partition, each with two sectors per cluster. Below is a FAT32 partition with a FAT size of three sectors followed by the first three data clusters. This illustrates how a FAT32 partition with the same cluster size can be created yet the data is no longer cluster aligned.

In the case of Windows, the file locations are available through standard API's, and the meta file contains itself as the first entry. In the case of Linux the NTFS driver does not provide a way to find out the clusters of a file. An ioctl was added to the driver for this purpose. A typical meta file is well under 8K in size so excessive memory use is not a concern.

Creating a meta file is not the only preparation required for formatting the NTFS partition as FAT32. The data files all reside on cluster boundaries. Unfortunately, NTFS numbers its clusters starting with zero at the first sector of the drive, while FAT32 begins its clusters at the sector immediately after the file allocation tables. Formatting with the same cluster size does not necessarily mean that the clusters will be aligned properly (see Figure 9).

A solution to the cluster alignment problem would be to always format the FAT32 partition with a cluster size of 512 bytes (one sector) and cluster downsizing the extent data by splitting it into 512 byte clusters. In practice this leads to an extremely large file allocation table when partitions run into the gigabytes.

The cluster size of the FAT32 file system is determined by constraining the size of the resulting file allocation table to be a configurable maximum size (default 32 MB). The simplest way to determine this is to loop over an increasing number of sectors per cluster in valid increments until the resulting calculation of the fat size exceeds the maximum. In order to align the clusters, we manipulate the number of reserved sectors until the newly created FAT32 partition and the former NTFS partition are cluster aligned.

At this point the layout of the FAT32 file system and the potentially larger cluster size is determined. Before formatting can occur, the extents of all the data files must be preprocessed to relocate any extent that is either located before the start of the FAT32 data area or does not start on a cluster boundary. In the best case, the cluster size has not changed, so only the first set of relocations must occur. Otherwise relocating an extent requires allocating free space on the disk at a cluster boundary and possibly stealing from the file's next extent if its length is not an integral number of clusters. Moving an extent's data is time consuming so it is avoided whenever possible. Free space on the disk is found using a sliding bitmap approach. Any cluster that is not in use by an entry in the meta file is considered free. A bitmap is used to mark which clusters are free and which are in use. The relocation process requires that enough free space is available to successfully relocate necessary portions of the files to be preserved. When restoring to the same partition this will always be the case.

Formatting is the simplest step. The mkdosfs program performs a semi-destructive format in that it only overwrites the reserved and file allocation table sectors. The '-f' switch is used to limit the number of file allocation tables to one.

Once the file system is formatted as FAT32, entries for the files to be preserved must be created. This is done via a user space FAT32 library written for this purpose. The user space library can mount a FAT32 partition and create directory entries in the root directory. It uses the data from the meta file to resurrect each

meta file entry by creating a directory entry and writing the extents to the file allocation table.

Once all of the files have been resurrected, it is safe to use the Linux FAT32 driver to write to the partition. The meta file is traversed once again to create the full paths and rename all of the files to their proper names. Finally, resident files are extracted from the meta file and written. At this point the partition has been converted from NTFS to FAT32 while preserving all of the files necessary to perform a restore.

Centralized Management

In the case of multiple work groups, management issues become highly important. If a system administrator is supposed to deal with multiple groups with ten or more PCs he will need some sort of an autonomic system to simplify the management of storage.

We based our system on IBMDirector which is widely available and boasts a high acceptance rate throughout the industry. To enable IBMDirector for our purposes we extend it in several ways. We developed extensions for the server, console and clients. Below we quickly detail the nature of those extensions.

Client side extensions are written in C++. The extensions provide all backup/restore functions as described elsewhere in this document. An important extension is related to communication between the client and server. The communication module relays all the requests and results between the two machines. The client also starts a simple web server which upon authorization provides information about the given client. This feature was implemented for the case where no IBMDirector server is available or when the server is not functioning properly. The information exported on the web page is the same as what can be obtained through the IBMDirector console. The information exported is shown below:

- workgroup name
- back-up targets
- date of last successful backup
- contact info
- Number of drives
- Size of drives
- Free space on each drive
- File system on each drive
- OS used
- Current status (performing back up, restoring, idle)
- User name and user info
- Location of the backup

Server and console side extensions are written in C++ and java. They are rather simple since all we need to add on the server are basic GUI elements that allow us to interface with the client and to receive data sent from the clients. The most complex extension is related to extending associations so that all StorageNet devices in the same work group appear in treelike form. The goal of this part of the project is to make a system that will be usable with or without the IBMDirector server.

The Corporate Environment

Our primary target environment in developing this system is a workgroup satellite office. If this is used in a corporate environment, there is the need for administrator level handling for setup, control and migration. Similar to the workgroup setting, the requirements of the workstation user are limited to:

- knowing my data is backed up (having confidence)
- knowing that my data is backed up to an area that will facilitate easy restoration

In contrast, the administrator in the corporate environment has requirements for additional control and data, including:

- wants different user's data to be distributed evenly (or specifically) across several servers
- wants reports specifying where a user's data is backed up and the usage per server
- during initial rollout, wants a way to seed the backup server destination to achieve the first goal
- during server migration, needs a way for the user's data to go to another server.

The general processing flow is described below.

The asset collection process on the user's machine sends the UUID (machine serial number) to an administrative web server. A long running process on the server discovers available backup targets. The administrator reviews a web page containing unassigned backup clients and discovered servers, and assigns these clients to a server. This information is recorded and used by the client backup process (usually scheduled) to keep the user's data. This assignment information is also used by the file and image restoration processes. Described graphically, we have:

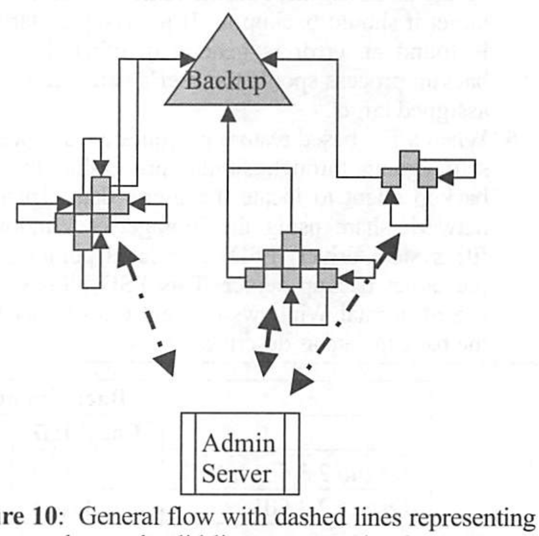


Figure 10: General flow with dashed lines representing meta-data and solid lines representing backup data.

The detailed processing flow is:

1. The asset collection process extracts the machine serial number, type, and location and uses HTTP POST to save this in a web server. If the asset collection process is disabled for this client, the user can surf to a well-known web page where the same executable from the asset collection process can be downloaded and run.
2. In parallel to this process, a long-running process resident on the server is busy discovering backup targets. These targets are StorageNet servers. The discovery protocol is limited to the subnet where discovery is issued. Because of this, there is a web service located at a well known address in each subnet that is used by the server-resident process to discover servers in other subnets. The list of available backup targets is maintained and updated in the administrative server.
3. The administrator surfs to a web page containing a list of unassigned clients and available servers. The processing behind this page automatically pre-selects target servers correlating to clients within their respective subnets. For those not pre-selected, or in which an override is requested, the administrator picks a server and one or more clients to back up to the selected server. This causes the machine file mentioned previously to be stored in that backup server. This is used for discovery by the restoration process.
4. The backup process on the client machine will normally be invoked as a result of a scheduled alarm "popping." When this occurs, the backup process will check for a machine file (containing its UUID) on all the servers on its subnet. If it finds this, it initiates the backup to that target.
5. If it does not find it, the backup process looks on the administrative server to determine which target it should backup to. If no assigned target is found an error is generated, otherwise the backup process spools the user's data out to the assigned target.
6. When a file-based restore is requested, a process starts going through similar processing to the backup client to locate the user's data. Then a network share using the StorageNet Windows file system driver (FSD) is created pointing to the target backup server. This FSD allows the use of normal Windows-resident tools to access the backup data as described above.
7. When an image-based restore is requested, a process starts going through similar processing to the backup client to locate the user's data. Then a network share using the StorageNet Linux file system driver is created pointing to the target backup server. This file system driver allows the use of normal Linux-resident tools to access the backup data as described above.

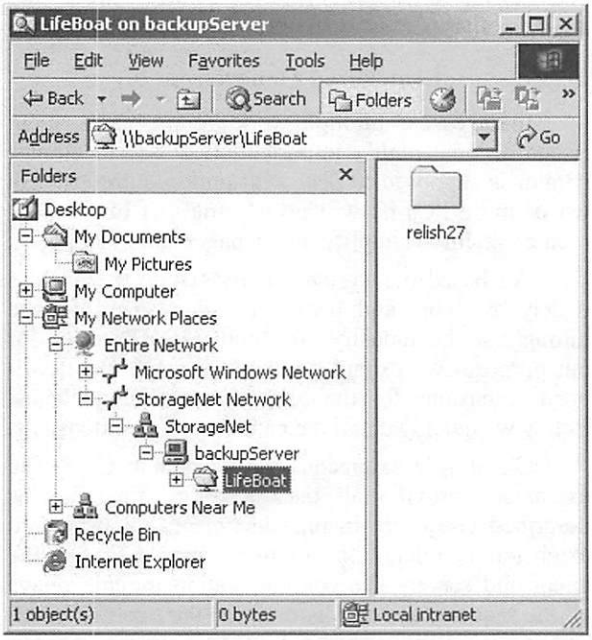


Figure 11: Image of the FSD accessing a StorageNet server.

Performance

During the extensive testing we gathered several interesting numbers that reflect the speed and efficiency of the backup and restore process [5]. Figure 12 shows the time in seconds for backing up and restoring 2.3 GB of data for a number of different target locations [7]. The restore process is measured from clicking on the restore button to the finish (reboot of a machine). Our main test machine is a ThinkPad R32 with 256 MB RAM and IBM 20 GB hard drive.

A separate series of tests were performed using a 1.6 GHz Pentium M IBM ThinkPad T-40. A 1.7 GB image requires three minutes (156 sec) to backup. The restore from local HDD requires 15.5 minutes from selecting the restore button of which ten minutes is file system preparation and data transfer and seven minutes is rebooting and converting.

Backup and Restore Times (seconds)				
	Local HD	**Local USB1.1**	**Local USB2.0**	**Remote 100 MB**
Backup 2.3 GB NTFS	808s	4254s	575s	1274s
Restore 2.3 GB	1100s	4440s	1001s	1200s

Figure 12: Backup and restore times in seconds.

The final stage requires two minutes to complete the attribute restore into the NTFS file system. This makes the total time 17.5 minutes. We have tested this repeatedly for at least 20 times on four systems with minimal variation. The main variations seem to be related to the file system preparation step which takes a minute or two longer after a base backup is re-established.

When compared to Xpoints software:

1. Backup is approximately 3X in performance.
2. Compression is typically 2X better.
3. Our version works without dominating the PC while Xpoint's version of RRPC does not.
4. The restore performance for a base only is similar.
5. Restore of an incremental plus base is dramatically improved in ours since it is essentially the same as a base only while Xpoint's takes about twice as long.

Conclusion

LifeBoat provides a way to backup system such that the backup files are accessible for single file restore as well as a full image restore. Our work also shows how Linux can be effectively used to restore a Windows(tm) system while also providing a rescue environment in which a customer can salvage recent files and preform basic diagnostics and productivity work. Most importantly this system allows for a machine to be completely restored from scratch when the boot disk is rendered unbootable.

The local backup version of this work shipped as part of IBM's Think

In this paper we presented a description of the latest research project in autonomic computing at IBM Almaden Research Center. We described a fully autonomic system for workgroup based workstation backup and recovery with options for both everyday restore of a limited number of files and directories as well as full catastrophe recovery.

This project is work in progress and is funded partially by the IBM Personal Systems Institute.

References

[1] Distributed Management Task Force, *System Management BIOS (SMBIOS) Reference Specification*, Version 2.3.4, December 2003.

[2] Reed, Benjamin C., Edward G. Chron, Randal C. Burns, and Darrell D. E. Long, "Authenticating Network-Attached Storage," *IEEE Micro*, Jan 2000.

[3] Gibson, Garth A., David F. Bagle, Khalil Amiri, Fay W. Chang, Eugene M. Einberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka, "File Server Scaling with Network-Attached Secure Disks," *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (Sigmetrics)*, June 1997.

[4] Miller, Ethan L., William E. Freeman, Darrell D. E. Long, and Benjamin C. Reed, "Strong Security for Network-Attached Storage," *FAST 02*.

[5] Zwicky, E. D., "Torture Testing Backup and Archive Programs," *Selected Papers in Network and System Administration*, USENIX.

[6] McMains, J. R., *Windows NT Backup and Recovery*, McGraw Hill, 1999.

[7] Stringfellow, S. and M. Klivansky, *Backup and Restore Practices for Enterprise*, Prentice Hall, 2000.

[8] Azagury, A., V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory, and L. Yerushalmi, "Towards an Object Store," *20th IEEE Symposium on Mass Storage Systems (MSST)*, 2003.

[9] http://www.trolltech.com/products/qt/ .

PatchMaker: A Physical Network Patch Manager Tool

Joseph R. Crouthamel, James M. Roberts, Christopher M. Sanchez, and Christopher J. Tengi – Princeton University

ABSTRACT

PatchMaker is a network management tool that tracks and stores descriptions of physical network patches in a database. Physical patches are cables that connect switches, patch panels, and room ports. PatchMaker allows administrators to manage switch ports from multiple vendors giving them the ability to enable/disable ports, make VLAN changes, and perform other switch port functions through a common web front-end. Finally, PatchMaker aids in host management by tracing network connections, identifying what kinds of devices are connected to the network, and helping to troubleshoot physical network problems.

Introduction

Over the past year, the network/systems administrative group in the Department of Computer Science at Princeton University has been putting together a plan (along with a good bit of code) to ease the lives of both the end-users and administrative staff. The many goals laid out by the team revolve not only around ease-of-use and happy end-users, but also around issues such as security, resource accounting (and, indeed, accountability), and administrative overhead.

PatchMaker was the first application of a suite of software. Our goal in developing PatchMaker was to integrate all the tools needed to deal with end-user requests related to the network into a single web GUI front-end. We wanted a system that would track physical patches, manage switch ports, track hosts, and be extensible. Since the cost of buying enough switch ports to connect all available ports on every wall box would exceed our budget, we needed a simple way to track patches from the wall boxes all the way back to switch ports.

In the past, whenever a user changed locations we would be required to move the original patch physically to another patch panel and, at the same time, to update the cabling documentation. We wanted to improve our turn-around time and respond faster to user requests. PatchMaker allows an administrator to locate a host quickly, find out what switch port it is located on, and move the cable to its new location without needing to trace the connection by hand.

The Department of Computer Science at Princeton University has 1600+ CAT 5e RJ-45 ports located throughout the building. Each room contains multimode and single-mode fiber optic ports along with RG6 for CATV. We are using two Foundry FastIron 1500 switches and an Alcatel OmniCore 5052 switch to connect the bulk of the user ports throughout the building. While most of the network connections "home run" back to a central machine room, we are starting to maintain our networks in other neighboring buildings, as we have run out of space in the original five-floor building.

Keeping track of all the connections from the room jacks, patch panels, and switches is very important, since users and hardware are moving all the time. Network administrators are constantly dealing with end-user moves, adds, and changes. Having out-of-date cabling documentation was always a problem and it increased the time required for an administrator to deal with end-user requests.

PatchMaker was built out of a need for a more automated way of resolving problems at the physical network layer. Prior to PatchMaker, system administrators manually entered changes reflecting patch additions, changes, and deletions into a database. The system was often out-of-date. As a result, administrators would eventually abandon the database altogether, forcing us to perform audits twice a year to re-inventory the location of all physical patches. In developing a patch database system that was easy to keep updated, and by adding tools to administer switches and to trace switch connections, we have found that our administrators are now using the system more effectively.

Motivation For the Project

Documentation for physical network cabling is essential in managing and troubleshooting a network. Documentation done after the fact is often wrong or out-of-date. In our previous system we would do a full building audit of the cabling and hardware connected to the network twice a year. We then would try to keep the data current using a locally written PERL script. Audits were very time consuming and often data was out-of-date even before the audit was even completed.

Once the information was stored in a database, administrators did not keep the patch information updated. The system was used to track patches, but

administrators had little incentive to use the system, because in most cases it was faster to trace a patch by hand than it was to use the database. Over the course of several years we attempted to institute procedures with the aim of encouraging administrators to maintain the system. We still failed to gain total compliance, as administrators typically respond to emergencies and to frequent experimentation by creating ad-hoc patches that never get documented.

As the department grows, new technology becomes available, systems administration tasks also grow, so we needed to identify areas of our work that could be automated, freeing up time to focus on other tasks. The PatchMaker system was developed to solve the problems we had with the previous system and to add functionality that would make the system more useful for other tasks related to network administration. The previous system's principle failing was that the data was not up-to-date. PatchMaker addresses this problem by making data updates fast and easy, while removing the previous ineffective system. The previous system only stored physical patches. PatchMaker adds more tools and bundles the functionality needed to complete the entire process in a single system, thus making it more useful.

We could not find existing tools that incorporated all the things we wanted. One open source tool, LANdb, did not fit our needs, as it only maintained a database of patches, and did not do switch management. The few commercial products that do exist are focused on switch management and tend to be very expensive. Other people we spoke with used a simple database, but they experienced similar problems keeping the data current.

While intelligent patch panels that totally automate the record keeping process exist, this was not a viable solution for us since we already had an infrastructure in place. It would have been expensive and time consuming to swap out patch panels. Intelligent patch panel systems usually log to a proprietary database. Proprietary databases are often inflexible, and using them makes it difficult to retrieve data for other purposes. Also, at the time we were considering them, intelligent patch panel manufacturers did not sell panels rated for Category 6 cable. This would have been a problem for us in the future.

What It Can Do

PatchMaker is a visual web GUI front-end to our network. Incorporating three main tools needed to make network changes, we created a tool that administrator's find useful and allows us to keep our cabling documentation up-to-date. The first part of the system, the network cabling documentation database, keeps track of where all the cables are located end-to-end in the building. The next part of the system is a switch management interface that allows an administrator to make changes on switch ports from multiple switch vendors. The last part of the system is host and device tracking and management.

PatchMaker allows an administrator to search for a host or device in multiple ways to find its location in the building. Using the built-in search tools, administrators can locate hosts by MAC addresses, hostnames, or physical locations such as room numbers or patch panels. Using a web interface an administrator can trace a patch, receive information about what is connected to a particular port, and make changes to whichever switch port the host is using.

When debugging a network problem, there are occasions when one suspects that the physical cabling or switch ports are not functioning properly. To help with this, PatchMaker will map the user's wall box location straight to the switch port and perform diagnostics remotely. PatchMaker is able to do this as the patch panel database contains mappings that go from the point the user plugs into the wall through intermediate patch panels to the end-point represented by a port on a switch. A user who is not connected is not generally happy. While simple solutions can sometimes be identified by the clients themselves, there are other times when cabling, or even a switch port can be faulty. In these cases, having the ability to map the user's wall box location straight to the switch port, and then remotely to perform diagnostics, is invaluable. PatchMaker does rely on a host being connected in order to locate it and gather information about the host. If a host moves to a port that is not physically wired to a switch port, there is no way of obtaining host information or doing configuration on the port. In this case, we would create a patch by hand.

In the course of normal network management, it is sometimes necessary to make small changes or to monitor ports: VLAN membership, link status, resource allocation (protocol, TCP/UDP usage, IP source and destination, bandwidth utilization). PatchMaker has the ability to configure ports from the switch-side and allows an administrator to enable/disable ports, change a port VLAN, or check the link status and speed of a port. This allows the administrator to quickly locate problems, identify hosts that may have a security problem, and disable the port to which a host is connected.

PatchMaker has a port-monitoring and alerts interface with links to graphs showing MRTG and Sflow (RFC 3176) data for the particular port. Using PatchMaker, we can access multiple tools from a single interface. For example, using SNMP and Sflow, we graph port traffic, maintain a log of traffic protocol data, detect sources of congestion, and manage quality of service on the switches.

Implementation

PatchMaker was developed using open source tools. It is written in PHP and DHTML. It uses PHP's database abstraction layer, courtesy of PEAR, and

allows for a variety of database types to be used on the backend. The database stores information about patches, buildings, racks, and rooms. It contains mappings that stretch from the point where the user plugs into the wall, through any intermediate patch-panels, to an end point represented by a port on a device. Switch information is stored in the database and includes attributes such as port count, device name, SNMP community strings, and switch slots. Image information for devices, patch panels, port pictures, and coordinates are stored in the database. An image map for a new patch panel or switch can be added and integrated into the new system in a matter of minutes.

The application runs very fast, querying the device, mostly via SNMP. The returned information is saved temporarily so other sections or panels with ports that patch to the same device can avoid re-querying the same information. It is written to work with various switches from different vendors, and plug-ins can be easily written for nonstandard devices.

The application displays a graphical representation of the patch-panels and devices to a web browser. The GUI lets the user search by building, rack, room, Ethernet address, hostname, etc. The user can select multiple patches and devices to view at the same time. At quick glance, the user can determine patch and link status by the color of the port image. Statistical information about each port, such as patch point, port speed, and VLAN can be viewed in the side-panel when the cursor is placed over it. Clicking on a port gives the user a context styled menu for enabling and disabling, changing VLANs, adding and deleting patches, changing rooms, as well as other commands.

PatchMaker incorporates links to other network monitoring tools which use Sflow (RFC 3176) and NetFlow to monitor our Foundry and Cisco switches, respectively, and gather port statistics. We also use SNMP to monitor traffic load on the switches and include links to MRTG (Multi Router Traffic Grapher) to display graphical images of the network traffic.

Even with some automation efforts, it is still necessary to enter patch information into the database. We have, nevertheless, built-in checks that watch for patches that have not been properly entered in the database. Figure 1 shows a device view in PatchMaker. In the figure, unidentified ports are marked with a black 'X' across the port. This allows an administrator to quickly identify ports that have patches, but are missing an end-point record. The system allows an administrator to know if a patch they are making is already defined in the database, and then it prompts them to update the database without having to remove the old patch. Figure 3 shows all the options that can be performed on a port. PatchMaker allows an administrator to perform switch port tasks like enable/disable ports, change VLAN, or port speed no

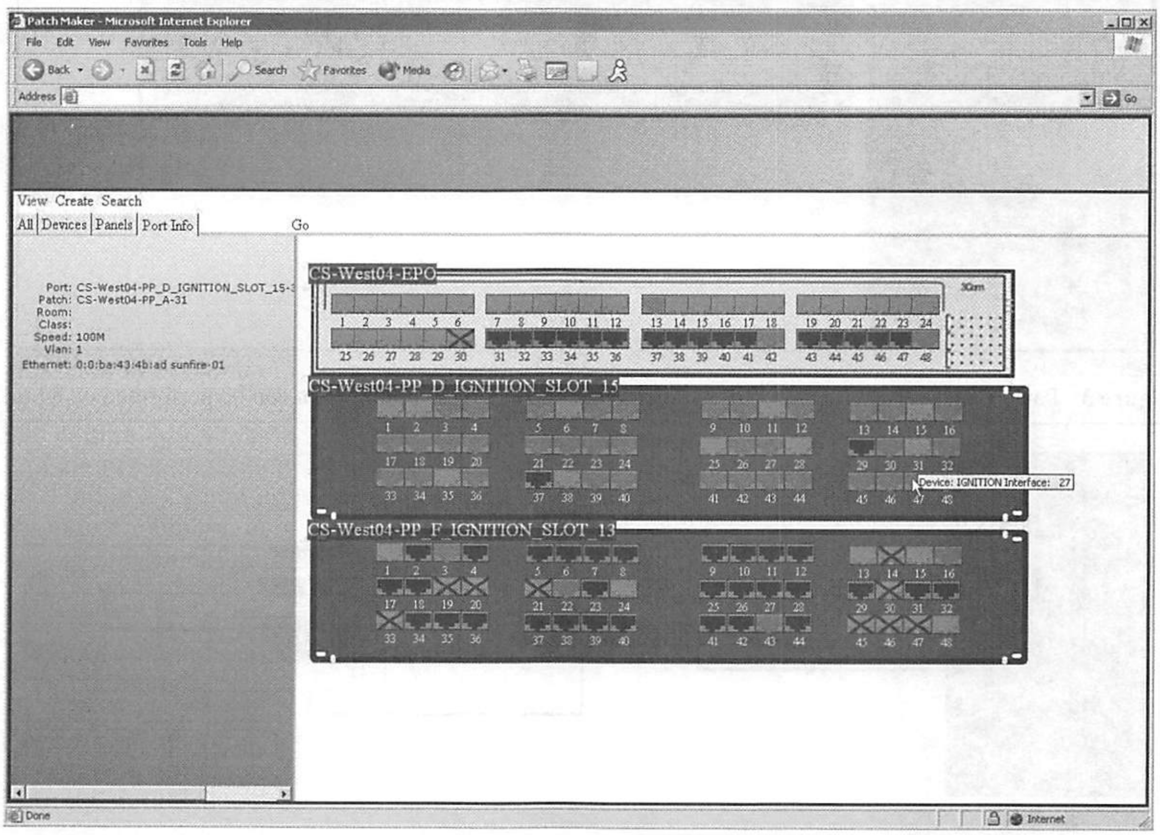


Figure 1: Devices view of PatchMaker.

matter what view you are in. Most network management tools give an administrator a switch view and allows for management of the switch from only this view. In PatchMaker if you view only the patch panels you can still change the switch port to which the patch panels connect to. Figure 4 shows a building room search and displays both the switch and patch panel information with an option to select a patch to view more information such as connected hosts. Figure 5 shows how you can display multiple devices and patch panels at the same time.

PatchMaker can handle more complex cabling infrastructures, including network closets and multiple cross connects, but Figure 2 illustrates a basic cabling diagram. In this diagram, a switch port is connected to a patch panel, then to a wall box, and finally to a host or other device. On the one hand, it is relatively simple to discover what host or device is on a particular switch port using switch management tools. On the other hand, information about which patch panel is used, and what wall box connections exist between the switch and the host, can only be discovered by hand tracing the connection or by keeping the patch documentation current. As long as patch panels and wall boxes are passive devices and do not signal their presence, this limitation is inherent.

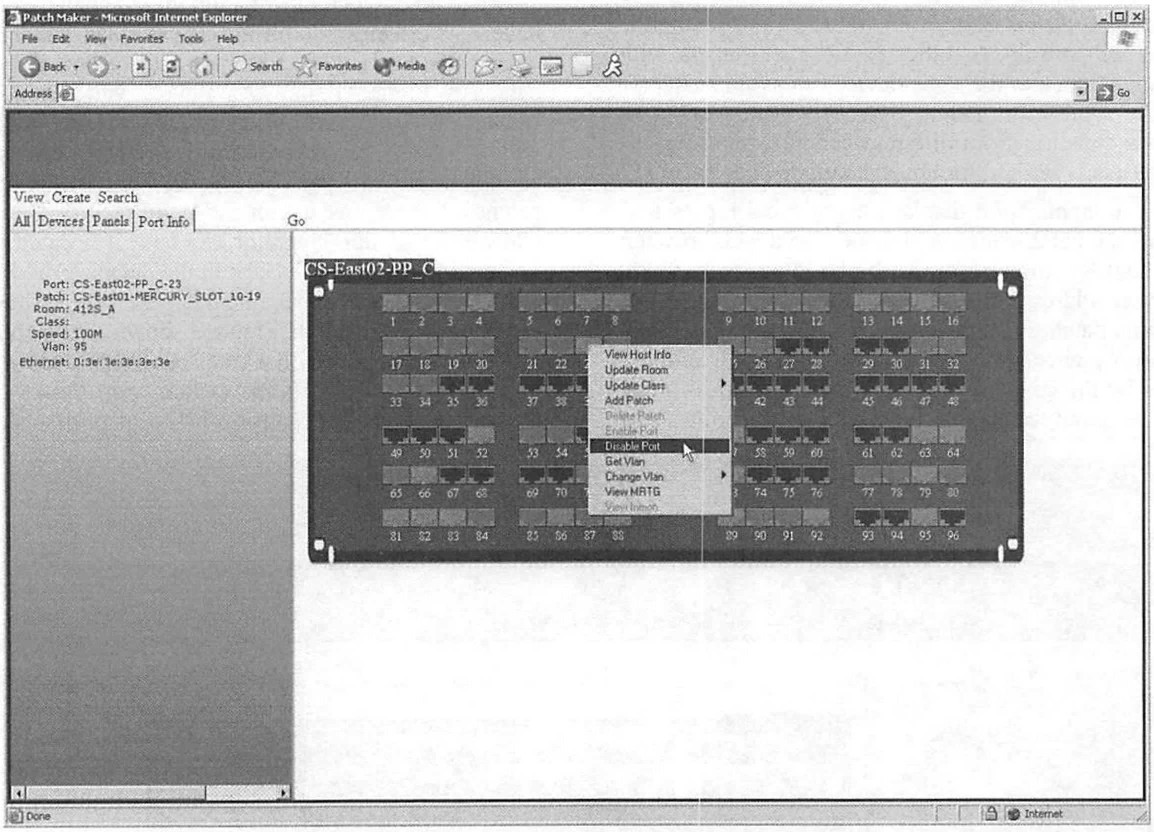


Figure 3: Patch Panel view with a drop down menu that shows all the options that can be performed on a port.

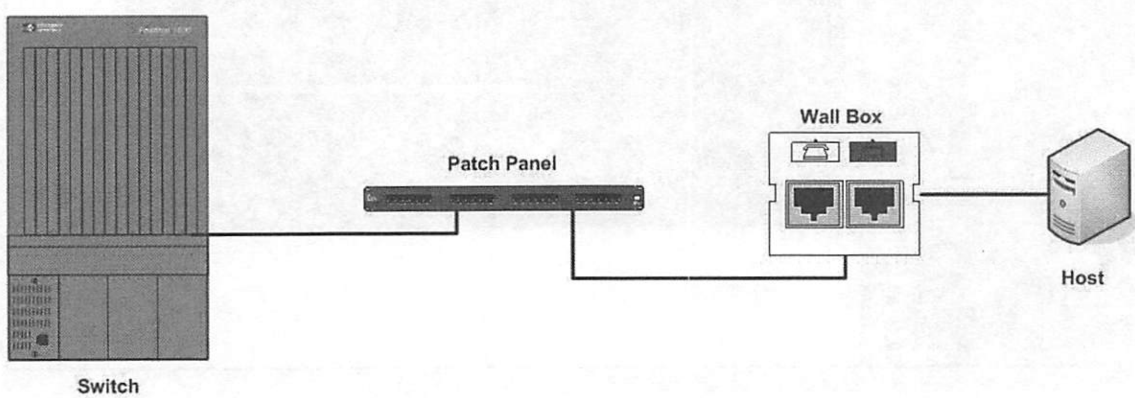


Figure 2: Basic network cabling diagram.

Future Enhancements

While PatchMaker is very useful to us, there are a few features we would like to add over time. In order for us to release this as a package so others can use it, we plan to write a GUI creation tool that would be used to set up patch panels and switch images in a drag and drop style with a small set of available images. We also would like to enhance user authentication to allow finer-grained control of user privilege levels. Other features we are working on include switch configuration file management, quality of service or rate limiting management, security access control list management, and change logging on the system and user levels.

Conclusions

The system was built out of a need for a more automated way of resolving problems at the physical network layer. Prior to PatchMaker, system administrators manually entered changes reflecting patch additions, changes, and deletions into a database. The system was always out-of-date. As a result, administrators would eventually abandon the database altogether, forcing us to perform audits twice a year to re-inventory the location of all physical patches. In developing a patch database system that was easy to keep updated, and by adding tools to administer switches and to trace switch connections, we found our administrators are using the system.

Since the system went into production, we have reduced user downtime and also reduced the time spent by administrators in dealing with network-related problems. We also decreased our response time in dealing with hosts that become compromised by a virus or trojan. We are able to disable the port and to take corrective actions on the hosts before the situation worsens. The search functionality also allows us to track hardware as it moves from room to room around the department. Our goal in creating the system was to build a tool that others could use and quickly deploy in their own environments.

Availability

PatchMaker will be released under GNU Public License (GPL) and will be made available at: http://www.cs.princeton.edu/patchmaker . To request more information about PatchMaker please send email to patchmaker@lists.cs.princeton.edu . We expect to release a public version of PatchMaker in November, 2004.

Acknowledgments

The authors would like to thank Chris M. Miller, Paul Lawson, and Craig Haley for their assistance in

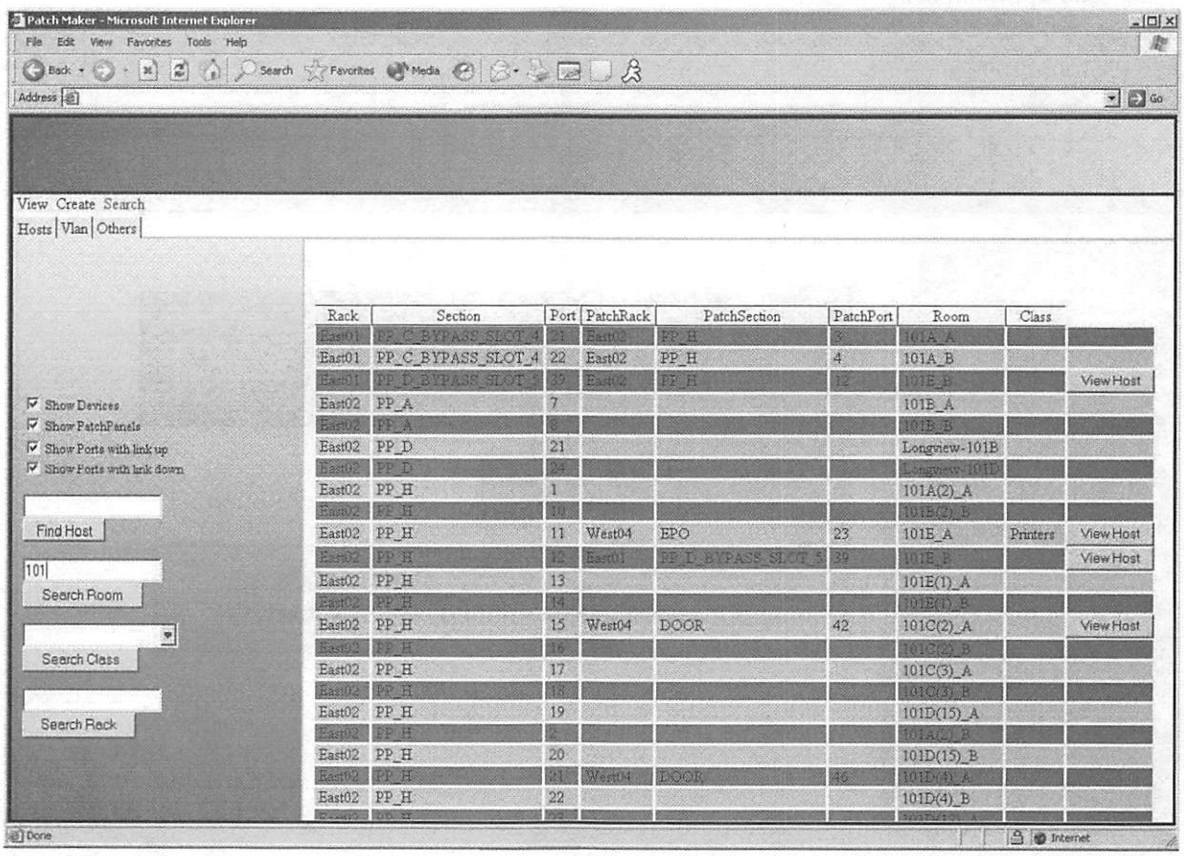


Figure 4: This image shows a search by room number. This shows the switch slot, port, and also the patch panel. You also can see more information about what is connected to a port by clicking the View Host button.

reviewing this paper. We would also like to express our sincere thanks to our shepherd Mario Obejas for his valuable feedback.

Author Biographies

All of the authors are members of the technical staff in the Computer Science Department at Princeton University.

Joseph R. Crouthamel is a System and Network Administrator in the Computer Science Department. He can be reached at jrc@CS.Princeton.EDU .

James M. Roberts is the Director of Computing School of Engineering at Tufts University. He can be reached at jmr@cs.tufts.edu .

Christopher M. Sanchez is a System and Network Administrator in the Computer Science Department. He can be reached at cmsanche@CS.Princeton.EDU .

Christopher J. Tengi is a System and Network Administrator in the Computer Science Department. He can be reached at tengi@CS.Princeton.EDU .

References

[1] http://www.usenix.org/publications/library/proceedings/lisa2000/full_papers/mitchell/mitchell_html/ .

[2] http://www.usenix.org/events/lisa2001/tech/full_papers/poynor/poynor_html/ .

[3] http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html .

[4] http://www.sflow.org/SamplingforSecurity.pdf .

[5] http://ucd-snmp.ucdavis.edu/ .

[6] Galvin, J. M., K. McCloghrie, and J. R. Davin, "Secure Management of SNMP Networks," *Proceedings of the IFIP TC6/WG 6.6 Second International Symposium on Integrated Network Management*, pp. 703-714, North-Holland, 1991.

[7] *Sflow*, http://www.sflow.org/ .

[8] Cisco, *Cisco Netflow Flowcollector*, http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc .

[9] Cisco, *Netflow Services and Applications White Paper*, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm .

[10] Dooley, Kevin, *Designing Large-Scale LANs*, O'Reilly, January, 2002.

[11] Mellquist, Peter Erik, "SNMP++ Specification," Hewlett-Packard Company, http://rosegarden.external.hp.com/snmp++/index.html .

[12] *LANdb: The Network Management Database*, http://landb.sourceforge.net/ .

[13] *MySQL*, http://www.mysql.com .

[14] *PHP*, http://www.php.net.

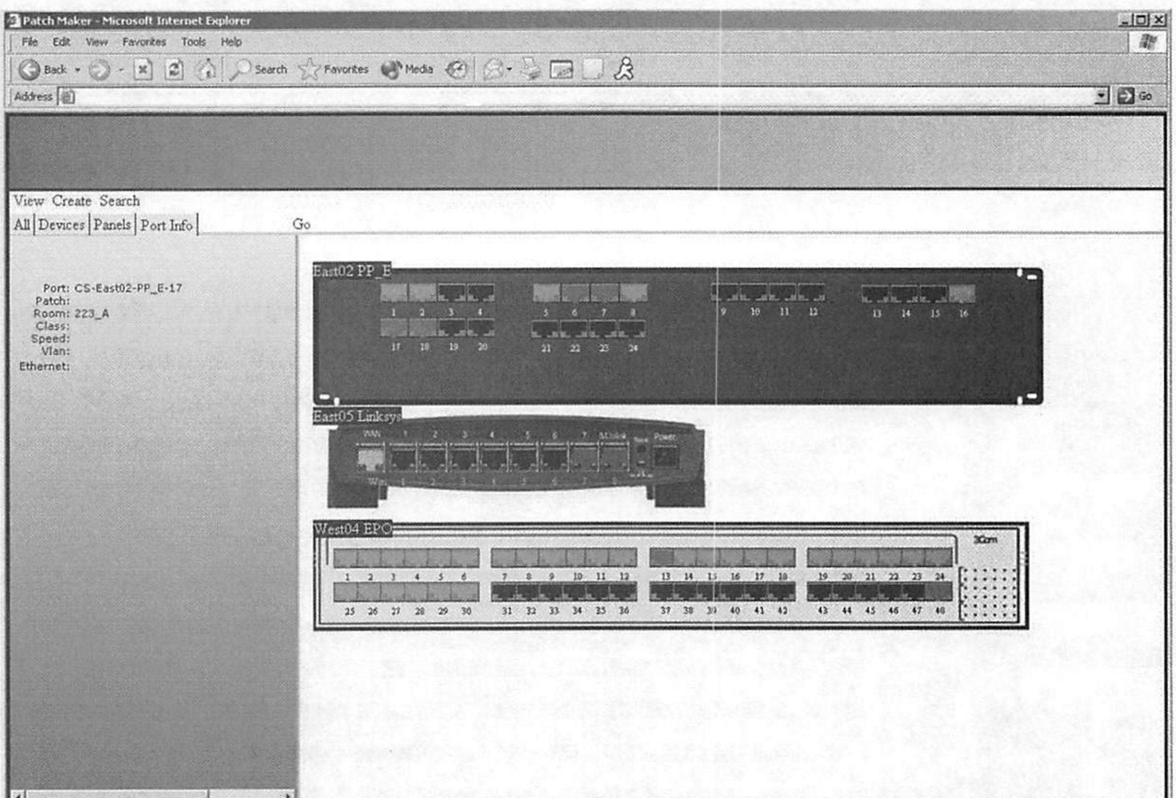


Figure 5: This image shows how you can display multiple things at once. Here we have a patch panel, LinkSys BEFSR81, and a 3Com 4300 switch.

Who Moved My Data? A Backup Tracking System for Dynamic Workstation Environments

Gregory Pluta, Larry Brumbaugh, William Yurcik, and Joseph Tucek – NCSA/University of Illinois

ABSTRACT

Periodic data backup is a system administration requirement that has changed as wireless machines have altered the fundamental structure of networks. These changes necessitate a complete rethinking of modern network backup strategies. The approaches of the 1980's and 1990's are no longer sufficient and must be updated. In addition to standard backup programs from vendors, specialized system administration tools are often needed. This paper examines one backup system and the major software components used to implement it. NCSA has developed a Backup Tracking System (BTS)[1] to perform backup operations based on knowledge of the network and when each machine was last successfully backed up. BTS can chronologically list all computers: from those currently attached to the network through those that have ever been attached over the life of the BTS program. BTS also provides information about all backup operations including the time of last attempt, success state, amount backed up, etc. The BTS database also contains the date of the last successful backup for each machine and whether it has at least one VIP user (to be given preferred status during backups) or all non-VIP users.

Introduction

Modern networks of end-user machines are becoming increasingly dynamic and heterogeneous. Operating systems come in various versions of Unix, Windows, or MacOS. Mobile hosts, which may only be available on the network rarely or on an intermittent basis, have become almost as common as desktop workstations. The data on individual hosts can be critical to the success of an organization (for cautionary stories of those who have been victims of data loss without backup see [19]).

A wide variety of backup and data integrity techniques exist, and they vary in cost, features, and effectiveness. Mirroring SAN systems are at the very high end. Such systems can provide real-time on and off-site mirroring and versioning of data as it is modified, and can allow quick recovery from both common and catastrophic failures. At the low end, users can individually manage backups to removable media, such as external hard drives, CD-R, or Zip disks.

While the techniques available to protect data have increased greatly, the management of such protection has not. Systems such as Amanda [14] expect collections of always-on, always-connected Unix workstations. Later commercial products like IBM's Tivoli Enterprise Management Suite [9] and Legato Systems NetWorker [11] have focused on extending support for the backend archive technology and new host operating systems.

[1]Funded in part by a grant from the Office of Naval Research (ONR) under the auspices of the Technology Research, Education, and Commercialization Center (TRECC) established at NCSA/University of Illinois.

Unlike previous systems, BTS is aware of the disconnected nature of modern networks and manages backup based on system and user priority. For example, a missed backup in most backup systems at best causes a host to be bumped in priority for the next scheduled backup. In contrast, BTS backups are on-demand, and if a host goes beyond the acceptable window without being backed up, a system administrator will be alerted to investigate the reason for failure.

Motivation

The differences in characteristics between older and more modern networks of end-user hosts necessitate revisiting the motivations and goals of data management. Modern networks are dynamic, and it is beyond the capability of current systems to cope with increasingly disconnected machines and backup latency.

The Reasons for Backups

There are many reasons why data backup is a crucial requirement for virtually every organization. The well-known, traditional reasons still hold. Disasters such as a flood and fire strike networks. Users inadvertently delete files and overwrite existing files. Hackers or disgruntled employees do the same intentionally. Disk drives, inherently fragile mechanical devices, fail, and lose all of the data they hold. Additionally, files become corrupted by bad disk sectors, magnetic fields, and improper system shutdown.

Beyond the traditional threats, there are new threats to today's systems. Thieves steal laptops, and the data contained on them, a threat which is much less applicable to traditional workstations and servers.

While the most skilled social engineer will have trouble convincing even naïve users to purposely delete data, even simplistic email viruses trick these same users into running hostile code with depressing regularity. Finally, the threat posed by modern worms dwarfs those of older worms [16], and they are able to compromise every vulnerable machine on the Internet faster than any manual response can prevent [17].

Organizations depend on their computer systems more than ever. Loss of data is therefore more expensive than ever in terms of lost work and downtime. Additionally, the public's increasing awareness of the importance of data security means that data loss has a large negative publicity component. With increasing threats and increasing costs, backups are more crucial than ever.

Lastly, we realize developing a backup strategy is an individual process specialized to specific network, data, and organizational objectives – different strategies work for different purposes. A survey of factors to consider such as contained in [8] provides an excellent planning tool for developing backup strategies.

Properties of Good Backups

In a well-managed network, backup operations are performed on a regular basis. Additionally a good recovery system is essential. During both normal use and recovery, backup operations should be transparent to users. Backup operations should be automatic and not be the responsibility of users. Instead, a system administrator should centrally manage backup and recovery operations. Since backups are a high priority, they should be managed by a person who understands their importance, rather than a new hire or intern. Finally, the scale of modern networks is beyond what can be manually managed. Good management requires human intelligence supported by automated information gathering and management.

Backup Nuances

Networks are categorized in various ways. A static network consists of physically attached workstations with a network structure that only administrators modify, and then only rarely. A dynamic network adds wireless machines and constantly changing physical structure. A homogeneous network consists of similar attached devices all running the same operating system. A heterogeneous network adds a mixture of various devices with different operating systems. Dynamic heterogeneous networks are a superset of static homogeneous networks, and performing backup operations in these networks is more complicated and requires additional tools. This paper discusses backup operations in the more general case, which applies to most modern networks.

We identify four distinct factors that account for backups being more complicated in a dynamic heterogeneous network. The proliferation of laptops exacerbates these factors, as laptops multiply the dynamism of the network.

- Networks consist of both physically connected and wireless computers. Both types need successful backups on a regular basis, yet each requires a different strategy. A physically attached workstation can be scheduled for backup when the network workload is light. A wireless computer requiring a backup must be scheduled while it is currently connected.
- Some computers may go days, weeks, or longer without logging onto the network. These machines cannot be backed up at a fixed scheduled time each night. To effectively backup these computers, a system must maintain information identifying the last time a computer logged on and the time of the last successful backup.
- Some users have multiple computers, such as several laptops and a workstation, which they periodically switch among. A person may use several machines in the same day and then use one machine exclusively for several months. All of the machines must be backed up.
- A machine can have multiple users who perform different types of processing. One user's job function may require preferred treatment of the machine during backups. Although some users are aware of the importance of backups, most are not and want no role in the backup process. Finally, some users' work habits are not conducive to good backup practices.

In a dynamic environment, the networked computers must initiate the backup operation, since the backup server does not know who is attached at a given time. Hence, software installed on each networked computer must coordinate data exchanges with the backup server. Whenever a new computer is added to the network, the backup client program should be part of the initial software load. Existing computers also need the client software installed.

Client software installation requires knowledge of which computers are actually present on the network. There may be no central point of control to identify when a new computer is added to the network. Likewise, existing computers can be permanently removed from the network without informing any authority. When a machine without backup software that has not been seen for months suddenly reappears, the machine's user needs to be contacted to install the software. Another computer not seen on the network for a comparable time may never reappear again. It would be a waste of time to contact its user.

Related Work

This section highlights backup systems or applied research with relevance to BTS. A comprehensive summary of all backup systems could not be included here, so we have selected a cross-section of the previous work. For a more comprehensive description of backup system issues and examples, see [8, 5].

Amanda (Advanced Maryland Automated Network Disk Archiver) is an early example of freely available backup management software [1, 14]. It uses a combination of full and incremental backups to concurrently backup networked clients to a single designated backup server and uses configuration files to determine the type of backup to perform. It has research significance in that it attempts to minimize cumulative overall backup per day in terms of number of backup runs, percentage change per backup run, and total amount of data [8]. Multiple commercial systems [6, 9, 11] now provide Amanda-like functionality; however, none deal gracefully with wireless hosts.

RAID [3] can protect systems against the failure of individual components. It provides no protection against unintentional/unauthorized modification of data, nor from catastrophic failure. Traditional RAID systems are impractical to field for mobile systems. However, a more recent RAID paper [15] applies RAID to a group of disconnected and distributed computers sharing storage remote from them. The aim is to provide a reliable RAID storage system that delivers acceptable performance while also providing a single coherent namespace for disconnected personal devices.

[4] proposes a taxonomy for backups, including categories such as full versus incremental, file versus device, online or not-in-use, snapshot, and copy-on-write etc. It then places well-known backup programs including xdump, tar, IBM ADSM, Legato Networker, Amanda, Plan9, and Andrew into the above categories.

Versioning file systems, such as Elephant [13], can protect against unintentional/unauthorized modification of data. However, a determined attacker can cause the history to be modified in undesirable ways. Even total versioning file systems like S4 [18] are no protection against physical failures.

[10] evaluates four backup algorithm strategies: (1) incremental, (2) daily-full, (3) mixture of full-incremental, and (4) concurrent backups using backup streams. The paper compares the efficiencies of these algorithms for both backup and restore operations.

In [7] a group of computers form a peer-to-peer network for backup operations. Data from one computer is distributed over other computers that have available capacity. The paper raises many non-standard backup issues related to confidentiality, integrity, authentication, and various other security issues. This is not currently a viable commercial solution but a very interesting paper nonetheless that may have future intranet applications.

The unique feature of BTS is its ability to prioritize backup based on system and user priority. The closest related work in the spirit of BTS is [2] which examines dependability in infrastructure systems by placing priority on components based on their utility in terms of economics and operations research. BTS carries this utility concept forward specifically as an ongoing backup process controllable by the user.

BTS System Description

To perform backup operations, a backup system must know which computers comprise the network and when each was last successfully backed up. Hence, in addition to the backup software and server, the BTS program monitors all networked computers and tracks backup status information, including the last successful backup date. BTS utilizes a database to manage this information on every computer that has connected to the NCSA network during the past several years. The relationship among these components is illustrated in Figure 1.

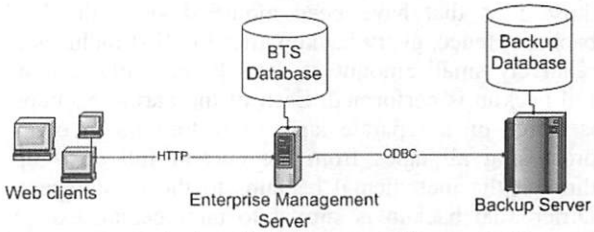


Figure 1: Relationship between Clients, BTS, and Backup Server.

BTS tracks whether users are logged-in via the NCSA authentication system and creates records of this activity at sixty-minute intervals. BTS also polls network machines to see if they are online. Then, for each online host, BTS uses algorithms based on system/user priority and time- since-backup to determine if an immediate backup is needed. If a backup is performed, the files are downloaded to a Mass Storage backup system containing 6 TB of disk cache and an ADIC tape library with six tape drives. An extensible database containing the specific information used to determine backup priority is used to coordinate information sharing for the different BTS components as well as for archiving and presentation to the user via a web interface. The data contained in the database includes the following: VIP users within the organizational hierarchy, problem machines, host-to-subnet mappings, and subnet-to-geographic location mappings.

BTS performs functions beyond helping with backup operations [12]. It provides information on network computers via the computer user name, IP address, and NetBIOS name. The Tivoli name is the identifier used for the backup processing. If the NetBIOS name and Tivoli node name are different, this will be indicated in the host list. BTS can chronologically list all computers, from those currently attached to the network through those that have ever been attached, over the lifetime of the BTS program. BTS also provides information about backups, including the time last performed, successful or unsuccessful, amount of data backed up, etc. Netview updates the BTS database every ten minutes. Hence only rarely will a network user go undetected. The BTS database also identifies each computer as having at least one

VIP user or only non-VIP users. Machines with VIP users have preferred status during backups. In practice, most computers have a single user categorized as a VIP or non-VIP.

Standard Types of Backup Operations

Historically three basic strategies have been used to perform backup operations, varying in the amount of data backed up and ease of restoration. A full backup backs up all data on a scheduled basis, and requires the most time and storage. However, it is the simplest to understand and the easiest from which to restore data. An incremental backup begins with a full backup. Subsequent backup operations copy only those files that have been modified since the last backup. Hence, every backup after the first includes a relatively small amount of data. Periodically, a new full backup is performed. Each of the partial backups is stored on a separate tape, so restoration involves processing all tapes from the current full copy up through the incremental backups to the most recent. Differential backup is similar to incremental, except that after the initial full backup, a single device is used for all of the incremental backups.

Progressive Backup Operations

None of the three basic strategies (full, incremental, differential) are well suited for a dynamic network environment since dynamics violate the timing considerations the standard techniques require. For example, a wireless user may only remain logged on for occasional short periods. To alleviate these issues a fourth backup strategy is used: Progressive Backup. Progressive backup initially copies all files on a computer and generates a summary report identifying when each file was last backed up and last modified. This report can also contain other file attributes such as size and creation date. The more information stored, the more efficient subsequent backup and recovery operations can be.

Each time a user logs on; a decision is made as to whether a backup operation is needed. Following the initial backup, subsequent progressive backup operations compare current file information with the summary report information. Based on this comparison, the backup only copies new and modified files. Unchanged files are not recopied. Determining the files that need to be backed up often requires more time than actually copying the data. Each backup operation also updates the summary report. Progressive backup can be extended to support versioning, where the most recent several versions and their summary information are saved.

During the backup, data is stored in the summary report relational database. SQL queries can be used to retrieve information about the backed up data associated with a given computer. Using the information about the backed up files in its database, it is possible for restore operations to be easily and correctly performed, something that does not always occur with incremental and differential backups. In some circumstances, these two strategies can restore redundant and even incorrect data.

Depending on the storage media, files copied during the current backup may not be contiguously stored with existing backed up files from the same computer. However, the backup systems' relational database identifies where each file from each computer can be located on the storage media. In this way, the database allows quick and easy restore operations to be performed.

In summary, progressive backups require less server time, minimize the required network bandwidth, utilize less storage media to hold backed up data, and are more accurate and efficient for restore operations than the other types of backups. When the host computer initially contacts the backup server, the server initiates a backup operation immediately for a laptop and schedules a later time for a workstation, typically after the workday ends.

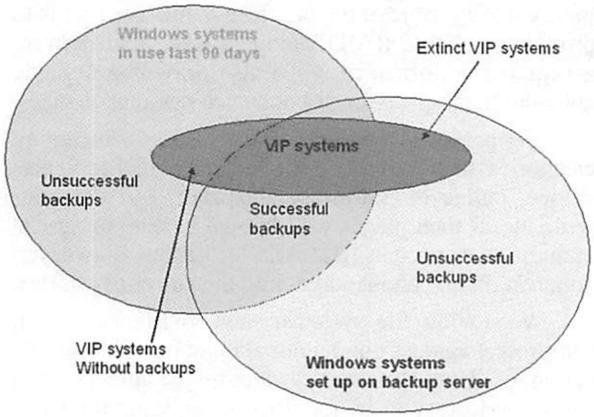


Figure 2: Venn Diagram categorizing backup status of clients.

Hierarchical Backup Strategy

Not all computers attached to a network need to have the same backup strategy – some computers and information are more important than others [2]. The Venn diagram in Figure 2 illustrates the various categories into which a computer can be placed. While performing progressive backups as described above, individual computers are prioritized, and time slots are assigned to each priority. Computers used by a VIP are considered more important and given more attention during backup operations than non-VIP computers. Part of this extra attention is currently provided manually, although the BTS program provides some help. The more important the VIP, the more important it is that timely backups are successfully performed on their machine(s). In the process of getting all existing users to install the client software on their computer, VIPs have been contacted first, in order of their importance. Computers used by the highest-level VIP are considered the most important computers. Computers used by VIPs reporting directly to this person are at the next highest level, etc. At the other extreme, the

users classified as least important will be contacted last. The system administrator responsible for backup operations generates this user importance ranking and implements a strategy for contacting the VIPs.

The manner in which a "VIP" is defined will depend on the way that makes most sense for a particular organization. For example, the most important VIPs may be one or more key software developers rather than the organization's president.

A backup failure on a VIP's computer is given priority. Such a failure is investigated and the reason for the failure identified with an entry in the BTS report of all VIPs whose system has not been backed up in the last 10 days. The BTS Reports shown in Figures 4, 5, and 6 illustrate that it is possible to determine which VIP machines have been successfully backed up.

Reports Produced By the Backup Tracking System

When the BTS program starts, it displays a Menu screen that allows several types of reports to be generated. Figure 3 shows the Menu screen prior to entering any data. Default values are provided for every data entry field including the three text fields.

The central portion of the Menu (labeled "Text Search") is used to generate a listing of all computers on the network whose name contains the value entered in the Find Host(s) by Netbios name; Find Host(s) by its NT user name or Find Host(s) by IP address. A partial wildcard may be entered in all three fields. With the IP address, a partial wildcard value is considered the beginning of the address. If nothing is entered, a listing of all computers in the network is generated. Figure 4 shows the results of entering PC in the user name field.

BTS generates reports that identify which machines have had a successful backup performed within the last N days, where N is 10, 30 or if it has ever been backed up. These reports can specify only machines belonging to VIPs, just non-VIPs or both groups. Three reports can be displayed that identify all computers successfully or unsuccessfully backed up within the last 10 days, the last 30 days and since the BTS started running.

Radio buttons on the right hand side of the menu screen allows a user to make selections in four categories to identify which computers will be included in the

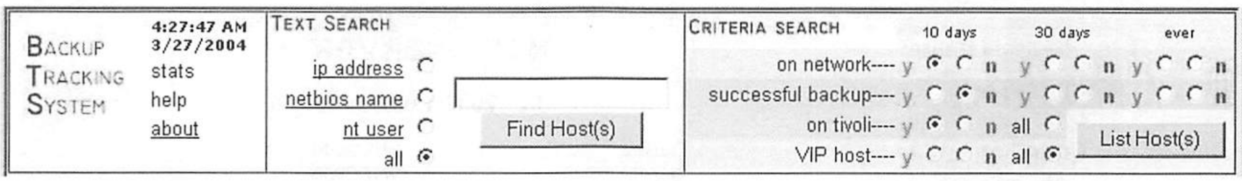


Figure 3: Menu screen showing default settings.

	HOST			BACKUP	
	NetBios Name * = Tivoli node name	location building(s)	Last in NCSA Domain	location building	date
1.	JIMMYPC	ACB / CAB	3/27/2004	ACB	3/6/2004
2.	JOHNPC	DCB / CAB	3/27/2004	CAB	3/8/2004
3.	JOEPC	CAB	3/27/2004	CAB	3/15/2004
4.	KARENPC	CAB	3/27/2004	CAB	3/5/2004
5.	ROBPC	CAB / CAB	3/19/2004	CAB	3/12/2004
6.	GREGPC	DCB	3/27/2004	DCB	3/10/2004
7.	MONITORPC	SRP / SRP	3/26/2004	SRP	3/1/2004

Figure 4: Listing of all computers whose user name contains "wildcardPC."

	HOST			BACKUP	
	NetBios Name * = Tivoli node name	location building(s)	Last in NCSA Domain	location building	date
1.	JIM-LAPTOP	SRP	8/2/2004	SRP	7/5/2004
2.	SUSAN-LAPTOP	SRP / ACCESS	8/2/2004	ACCESS	6/25/2004
3.	JAMES-DESKTOP	CAB / BI	7/26/2004	BI	7/19/2004
4.	STUDENT1	BI	8/2/2004	BI	7/21/2004
5.	RESEARCH-SERVER	CAB	8/2/2004	CAB	7/23/2004
6.	WEB-SERVER	Offsite / Offsite	8/2/2004	SRP	7/21/2004
7.	BACKUP-SERVER	SRP / ACB	8/2/2004	SRP	6/21/2004
8.	TEST-SERVER	SRP / ACB	8/2/2004	SRP	6/21/2004

Figure 5: Listing of all computers meeting a specified criteria.

report. One of three choices is selected from VIP Host, Non-VIP Host or Both Categories. One of two choices is selected from Contains Client Software (Tivoli) or does not contain Client Software. One of two choices is selected from Successfully or Unsuccessfully backed up. One of two choices is selected from On or Off the network. For the previous two choices, an additional selection is made from one of three time intervals: last 10 days, last 30 days, never. Figure 5 shows the results of entering VIP host, Client Software Installed, and Unsuccessful backup during last 10 days

Any of the computers in the Figure 5 listing can be selected to have BTS generate a report providing additional general information and the status of recent backup operations for that machine. Figure 6 shows a report of the most recent backup operation results for computer NCSA-SERVER.

Unsuccessful Backup Operations

There are several reasons why some machines are not successfully backed up. Most commonly, the computer does not have a backup client installed on it. BTS is used to identify these machines. To resolve this problem with existing networked computers, it is necessary to contact the user of the machine and install the software, a time consuming process.

Another possible reason for failure is that the system has a backup client and is part of the network, but is unavailable to backup. BTS also identifies these machines. Many users have multiple machines and are currently using only one of them. If all the machines are not being used simultaneously, the machines not being used are not being backed up because they cannot be accessed. If a workstation is powered off at the end of the day, it cannot be backed up that night.

Failure can also occur when the backup client software is installed on the client, but is incorrectly configured on the server. The server must be scheduled correctly in order to backup the client computer.

Finally, a few backups may inexplicably fail and require a restart of the backup server.

Vendor Product Issues

There are several commercial products that can do the type of processing described in the Progressive

NetBios Name:	NCSA-SERVER	
last seen in Network Neighborhood:	3/27/2004 1:00:04 PM	
Authenticated Windows user(s):	sumike sujim susue suamy Administrator surobert	3/19/2004 3/17/2004 3/5/2004 2/19/2004 2/19/2004 1/9/2004
Recent IP Number(s):	124.126.28.39 124.126.28.50	ACB High-end systems ACB Production Servers

Tivoli Activity Log

		OBJECTS			TIME	
date/time	bytes	inspected	backed up	failed	transfer	total
3/27/2004 3:45:27 AM	1.38 GB	1,385,363	1,193	45	4 min	02:30:14
3/26/2004 3:43:24 AM	613.93 MB	1,384,646	1,049	46	2 min	02:28:02
3/25/2004 11:07:56 AM	940.81 MB	1,383,900	585	22	3 min	01:21:22
3/24/2004 3:47:38 AM	1.02 GB	1,383,733	494	22	3 min	02:32:08
3/23/2004 5:13:40 AM	12.81 GB	1,383,574	3,943	22	90 min	03:58:09
3/22/2004 3:35:57 AM	88.88 MB	1,381,592	83		1 min	02:20:49
9/24/2003 4:12:01 PM	Tivoli Installed					

Figure 6: Report on backup information For computer NCSA-SERVER.

Backup Operations section including the Tivoli Enterprise Management Suite [9] from IBM, Retrospect [6] from Dantz Development Corp. and Networker [11] from Legato Systems.

However, some backup products are designed specifically for small networks and do not scale to a network the size of NCSA. At NCSA, Tivoli is used to perform the actual progressive backup operations. Tivoli performs all of the relevant backup processing and does not significantly interfere with regular network traffic. Several other products not mentioned were initially tried but they negatively impacted network traffic. In addition, with some earlier products the server initiated the backup rather than the client, a bad idea in a dynamic networking environment.

The Dantz Retrospect Professional product is designed for home and small offices using Windows and Apple Macintosh computers [6]. It does progressive backups with 100% correct restores and no redundancy (a significant feature). It uses compression and can backup data to any media. However this solution does not scale to larger networks the size of NCSA.

Other products can simultaneously backup dozens of clients. When the client contacts the server to determine whether a backup is needed, the server makes a decision based on the identity of the client computer. Criteria can include the following: the client is a wireless with a VIP user – backup immediately; the client is a server or a workstation belonging to an important VIP – backup every 24 hours; a user workstation – backup every weekday, but not over the weekend; and a VIPs computer not seen on the network for weeks – backup immediately.

Another significant backup issue is how to process the files that comprise well-known applications that are running on most computers. Examples include Microsoft Word, Excel, Access and even the operating system itself. It should not be necessary to copy these files from almost every machine. The backup software can be provided with a list of files to exclude during backups. Alternatively, application software and the operating system can be reinstalled rather than restoring from a backup.

BTS consists of an ASP application written in VBScript running with Microsoft IIS 5.0 on a Windows 2000 server. BTS uses an Access database containing information about the networked computers. The database is distinct from the relational database used by commercial backup software such as Tivoli.

Availability

Various statistics have been collected from the NCSA network, but the most interesting and valuable have been measurements of availability in terms of systems and users.

Let s represent the number of systems on the network at a given point in time, measured every ten minutes. Therefore, the normalized system and variation for a given time period can be respectively defined as:

$$N_s = \frac{\sigma_s}{\bar{s}} \qquad N_u = \frac{\sigma_u}{\bar{u}}$$

In an environment which is extremely static, and therefore the systems are on the network and each user logs in every work day, $N_s = 0$ and $N_u = 0$. As the number of systems and users per day varies, the values of N_s and N_u increase. For example, if the number of users vary an average of $\pm 10\%$, then $N_u = 0.10$.

$$\bar{s} = \frac{1}{n}(s_1 + s_2 + s_3 + \cdots + s_n)$$

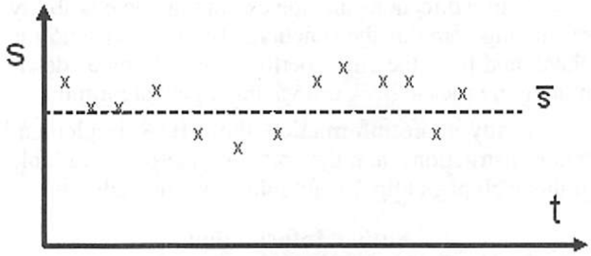


Figure 7: Hypothetical system availability.

Similarly, let u represent the number of users who authenticate on a given 24-hour period, recorded once per day at midnight.

$$\bar{u} = \frac{1}{n}(u_1 + u_2 + u_3 + \cdots + u_n)$$

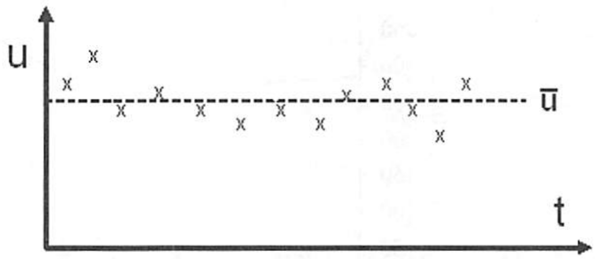


Figure 8: Hypothetical user availability.

It should be expected in environments which have high values of N_s and N_u that it would be considerably more difficult to backup, apply security patches, and track down systems than in an environment with lower values of N_s and N_u. High values of N_s and N_u would therefore imply a higher security risk for a given infrastructure effort, or to put it another way, a higher support cost for a given level of security and survivability. This ability to track usage trends has proved useful for capacity provisioning, security events, and equipment reliability failures.

Figure 9 is a sample of real measurements of system and user availability (respectively) on the NCSA network. Over a three-year period of measurement, the average number of systems available is 300 with 600 distinct systems, an upper limit of 400, and a lower limit of 100. The normalized system variation is 0.10. Over the same three-year period of measurement, the average number of users is about 190 users with 220 as the upper limit and 30 as the lower limit. The normalized user variation is 0.53.

Conclusions

Sharing the general class of backup problems we face at NCSA and our specific implementation solutions have proved to be valuable to peer organizations. We feel that the solutions we describe in this work are transferable to other environments even though this work was specifically targeted to the Windows environment. In fact, we already have a parallel project in progress transferring these same techniques to the Linux environment.

Future directions include examining the possibility of moving some of the functionality of Tivoli onto the client, and have the client perform tasks (such as determining the files to back up) via intelligent algorithms.

Lastly, more information about BTS, implementation instructions, and the software itself are available at this web page http://wegpublic.ncsa.uiuc.edu/bts .

Author Information

Gregory C. Pluta is currently Manager of the Windows Environment Group at NCSA. He is a graduate of the University of Illinois at Urbana Champaign (Aerospace Engineering, BS 1992, MS 1995). As a graduate student, Greg was responsible for the department's student computer scientific workstations, designed and built a nonlinear systems laboratory, and re-designed undergraduate engineering laboratories with modern data acquisition and analysis systems which he taught for two years. Greg then worked at Andersen Consulting as an analyst writing business and multimedia software for two years on more than a dozen different projects in almost as many languages, and as a software configuration management specialist for large software development projects. Greg then joined NCSA as a system engineer, then as manager of Windows desktop and server systems. Greg can be reached at gpluta@ncsa. uiuc.edu .

Larry J. Brumbaugh was born in Pittsburgh, Pennsylvania and graduated from the University of Pittsburgh with a BS in Mathematics in 1965. He obtained an MA in Mathematics from West Virginia University in 1968 and an MS in Computer Science from the University of Kentucky in 1976. He is ABD in Mathematics (Kentucky) and Computer Science (University of Illinois). In a long and varied career, he taught Computer Science for three years at Morehead State University in Kentucky and for 23 years at Illinois State University in Normal, Illinois. He is the author of two computer science books and numerous papers and presentations. His teaching and research

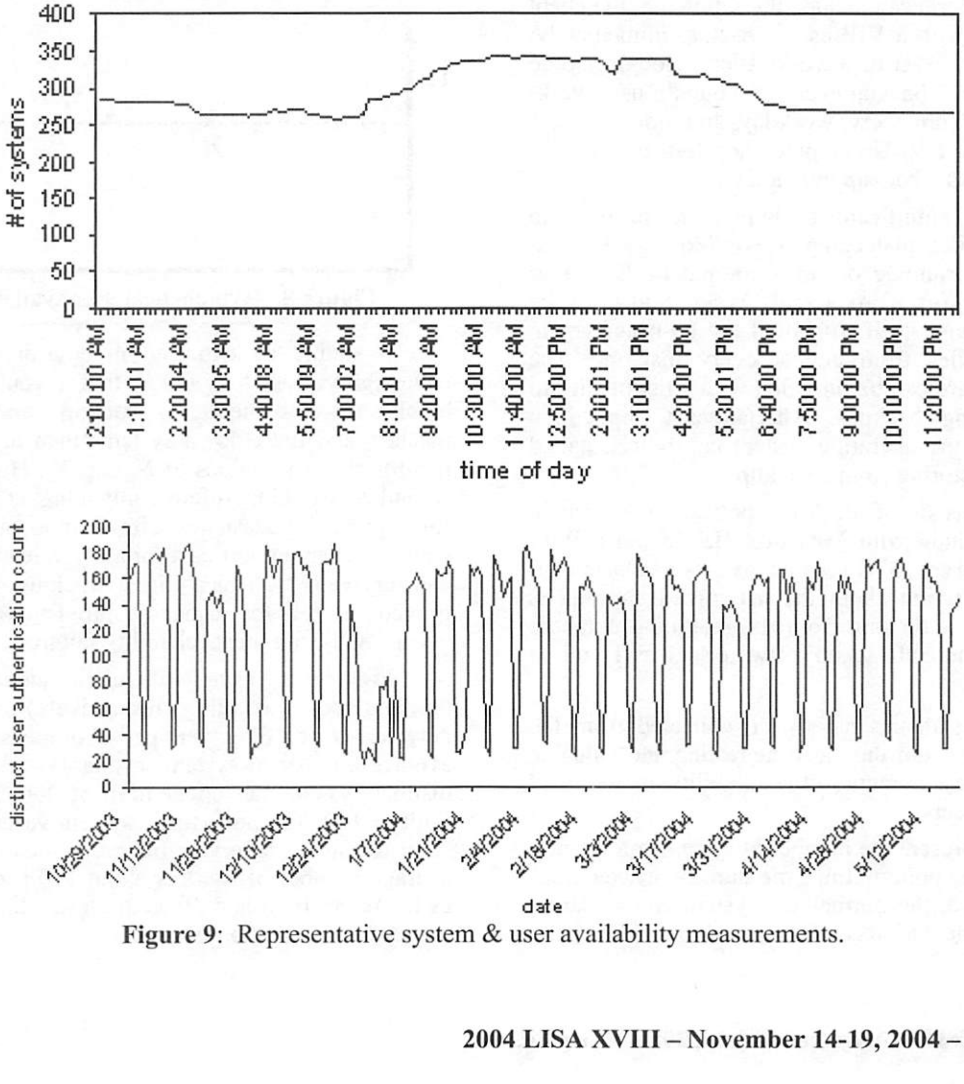


Figure 9: Representative system & user availability measurements.

interests have included mainframe programming, data communications and networking and computer architecture. He has worked as a consultant for many companies and government agencies. He joined NCSA in February 2004 as a consultant on Storage Security. Larry can be reached at ljbrumb@ncsa.uiuc.edu .

William (Bill) Yurcik is currently Manager of Security Research at NCSA. Priorto this he was Manager of Security Operations at NCSA, so he has both a theoretical and practical background in computer network security. He is a graduate of Johns Hopkins University (MS Electrical Engineering 1990, MS Computer Science 1987), the University of Maryland (BS Electrical Engineering 1984), and is ABD from the University of Pittsburgh (1994-99). He had 12 years of professional experience as a Network Engineer prior to joining NCSA (NRL, NASA, Verizon, MITRE). He has also been a Visiting Professor at Illinois State and Illinois Wesleyan Universities for three years, and since 2001 is an adjunct Professor of Computer Science at the University of Maryland. Bill can be reached at byurcik@ ncsa.uiuc.edu .

Joseph Tucek graduated from Washington University in St. Louis in 2003 with a BS in Computer Science and a BS in Computer Engineering. Although he played around with AI and robotics, he has found his true interest in systems. Joseph is currently a Ph.D. candidate in Computer Science at the University of Illinois at Urbana/Champaign, working in the Storage Security group at NCSA for support. Joe can be reached at tucek@ncsa.uiuc.edu .

References

[1] *The AMANDA Homepage*, http://www.amanda.org .

[2] Candea, George and Armando Fox, "A Utility-Centered Approach to Building Dependable Infrastructure Services," *Tenth ACM SIGOPS European Workshop (EW)*, September 2002.

[3] Chen, Peter M., Edward L. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, June 1994.

[4] Chervenak, Ann L., Vivekanand Vellanki, and Zachary Kurmas, "Protecting File Systems: A Survey of Backup Techniques," *Joint NASA and IEEE Mass Storage Conference*, 1998.

[5] Preston, W. Curtis, *Unix Backup and Recovery*, O'Reilly and Associates, 1999.

[6] Dantz, *Dantz Retrospect – Intelligent Backup and Restore*, http://www.nwfusion.com/whitepapers/dantz/whitepaper.html , June 2004.

[7] Elnikety, Sameh, Mark Lillibridge, Mike Burrows, and Willy Zwaenepoel, "Cooperative Internet Backup Schemes," *Usenix Annual Technical Conference*, June 2003.

[8] Frisch, Æleen, *Essential System Administration Third Edition*, O'Reilly & Associates, 2002.

[9] IBM Software, *IBM Storage Management Solutions*, http://www.nasi.com/tivoli_backup recovery.htm , 2004.

[10] Kurmas, Zachary and Ann L. Chervenak, "Evaluating Backup Algorithms," *IEEE Symposium on Mass Storage Systems*, 2000.

[11] Legato Software, *Legato Networker*, http://www.legato.com/products/networker/ .

[12] Pluta, Gregory, Larry Brumbaugh, and William Yurcik, "BEASTS: An Enterprise Management Tool for Providing Information Survivability in Dynamic Heterogeneous Networked Environments," *IEEE Local Computer Networks Conferences (LCN)*, November 2004.

[13] Santry, Douglas S., Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir, "Deciding When to Forget in the Elephant File *System," Symposium on Operating Systems Principles (SOSP)*, December 1999.

[14] da Silva, J., and O. Guomundsson, "The Amanda Network Backup Manager," *Proceedings of the Seventh Large Installation Systems Administration Conference (LISA)*, November 1993.

[15] Sobti, Sumeet, et al., "PersonalRAID: Mobile Storage for Distributed and Disconnected Computers," *Usenix Conference on File and Storage Technologies (FAST)*, January 2002.

[16] Spafford, Eugene H., "An Analysis of the Internet Worm," *Proc. European Software Engineering Conference*, September 1989.

[17] Staniford, Stuart, Vern Paxson, and Nicholas Weaver, "How to Own the Internet in Your Spare Time," *USENIX Security Symposium*, August 2002.

[18] Strunk, John D., Garth Goodson, Michael L. Scheinholtz, Craig A. N. Soules, and Gregory Ganger, "Self-Securing Storage: Protecting Data in Compromised Systems," *Fourth Symposium on Operating Systems Design and Implementation (OSDI)*, October 2000.

[19] *Tao of Backup Wailing Wall Homepage*, http://www.taobackup.com/wailing.cgi , June 2004.

Making a Game of Network Security

Marc Dougherty – Northeastern University

ABSTRACT

This paper describes our experiences in the design and implementation of a network for security training exercises, and one such exercise. The network described herein is flexible, and could be used for a wide variety of training exercises. Organizations of all sizes can use this type of exercise to train new personnel or keep existing staff at their best. In addition to the training benefits, these exercises can also be highly entertaining. The designs described here are intended to assist others in the construction of similar networks, and additional training scenarios.

Introduction

Companies large and small invest a great deal of money in the training of security personnel and system administrators. However, there is little formal training available for individuals without a company willing to pick up the check. This leaves the majority of sysadmins to learn the old fashioned way: by making mistakes. This is especially true of aspiring sysadmins at the college level, where system administration and security are discussed as a side note, if at all. In light of this, we set out to create a safe environment where all types of sysadmins could practice their skills, without putting personal or business data at risk.

Inspired by the "Capture the Flag" competition held annually at Defcon [1], we sought to create a similar environment, with particular emphasis on security and teamwork. Security has become a higher priority for many companies, security-related spending has increased drastically, and as a result, many sysadmins have been "promoted" to specialized security roles, without any additional training. Teamwork is also a common weakness for sysadmins. Many sysadmins work individually, and few have much experience working with others. Team training exercises like those proposed here can help overcome these common weaknesses and be entertaining at the same time.

Because of the sysadmin predilection for experiential learning, informal training exercises like those described here can be enormously beneficial, both for the individual sysadmin and his or her employer.

The environment described here is the result of hard work from a group of students at Northeastern University, working jointly with the Systems Group at the College of Computer and Information Science.

The network we have created involves two smaller networks, connected by an OpenBSD machine which performs the majority of the routing, and provides limited access to the Internet from both networks via HTTP, HTTPS and SSH. Internet access is vital as it allows access to the most current patches, tools, and technologies for both defense and offense.

The first internal network, called the defender network, is populated with the machines that contestants administer. We provide contestants with a machine on this network, which they must defend. Contestant teams may choose their preferred OS from a set of alternatives provided by the contest admins. Members of the administrative team often sabotage default security features of these installations, making the defender's job more interesting.

The second network is a new addition to the contest, inspired by many requests of participants to bring along their own machines for use in the contest. The attacker network was created to allow participants to plug into the contest, while minimizing their risk of being attacked. A strict set of ACLs prevent incoming connections to machines on this network and prevent hosts on this network from communicating with each other.

The keystone of the network is the routing policy on the OpenBSD gateway machine. This routing policy is written entirely in pf, the OpenBSD packet filter. No incoming connections are allowed to the attacker network, but responses to existing connections are allowed. All traffic from the attacker network to the defender network is passed through unmodified, because much of that traffic will be crafted packets and exploits which should not be tampered with. Other duties of this gateway machine include serving DHCP to the attacker network and providing DNS, NAT, and Internet access for both internal networks. This machine is also responsible for the Nessus server which is used by the scoring system.

Since this scenario was designed to be as realistic as possible, there is a minimal rule set. The only form of attack that is disallowed outright is denial of service attacks. However, the rules allow the contest admins to deem the conduct of a team "unsportsmanlike," and punish them accordingly. Punishments can range from a score penalty to expulsion from the contest.

The total score of each team is the sum of offensive and defensive scores. The offensive score is very difficult to automate, and has typically been handled manually. When a participant compromises a service, they are required to leave some type proof, which the contest admins will then verify. Defensive scoring is

based on the availability of the team's services. Periodically, the scoring system tests the functionality of each service on the contestant's host using several custom Nessus plugins written for this purpose. The state of these services is then recorded in a database. From the database, the information can take on any number of forms for presentation. Previous methods have included a text file, a web page, and an XML document processed by a Macromedia Flash application which displays the information in a more exciting way.

This contest has evolved to its current condition over the last several years, and continues to do so. Work is being done to move all routing policy into a Cisco Catalyst switch, which frees up the OpenBSD gateway machine to act as a web and ftp proxy for outbound connections. Also under development is a fully automated scoring system which merges the offensive and defensive aspects of scoring. The new system uses flag files to indicate when a service has been compromised. This scoring system requires significantly less manual intervention, and simplifies the lives of the contest administrators.

Because we have been unable to find a suitable metric by which to measure system administration skill, we cannot prove that the participants in our contest have become better admins. However, many participants have reported learning a great deal from the contest. In addition, many former participants have become members of the contest's administrative team, which indicates that the contest inspires participants to strengthen their teamwork and security skills.

Network Layout

Designing a network for use in security training involves finding a delicate balance in communication policy, to assure that participants are given enough freedom to use various attack techniques, without granting them the ability to attack machines outside of the scope of the training environment. The environment we have created consists of a defender network segment, and an attacker network segment. These networks are connected to each other, and to the Internet, by a gateway machine that performs routing and packet filtering.

Access to the Internet is provided to allow participants to research protocols and programs with which they are not familiar. The defender network is populated with machines created by the contest administrators. The attacker network must be able to attack the machines on the defender network, while remaining protected from the attacks of others. Both networks must be prevented from launching attacks on other machines outside of the training environment. The establishment of proper routing policies is critical to the success of this network. If the routing policies are not restrictive enough, this network could be used to launch attacks to the outside. On the other hand, if the policies are too restrictive, they may interfere with the intended use of the network by blocking some types of anomalous traffic.

Internet Access

The simplest of the network policies involved is the routing for the Internet connection. Both the attacker and defender networks are permitted access to the Internet via HTTP(S) and SSH. All other traffic bound for the Internet is dropped. In the past, we have allowed teams to participate remotely by forwarding external ports on the gateway to hosts on the defender network. Each machine on the defender network is accessible by Remote Desktop and Secure Shell. In the most recent incarnation of the contest, we encouraged participants to physically attend, eliminating the need to provide remote access.

Defender Network

The defender network is home to the machines that the contestants must defend. In order to give participants the opportunity to experiment with sniffing tools, the defender network has typically been hubbed rather than switched. This network uses private, non-routable IP addresses in the 10.10.0.0/24 range [2], which are provided by the gateway. To decrease the potential for abuse of this network, Internet access is limited to HTTP(S) and SSH. To minimize the vulnerability of machines on the attacker networks, defender machines may not initiate connections to the attacker network.

Attacker Network

The attacker network was a new addition in the most recent incarnation of the network and its design posed an interesting challenge. The goal of this network was to allow participants to bring along their own machines for use the attacking portion of the exercise, while minimizing the risk associated with plugging into such a hostile network.

The attacker network uses private non-routable IP addresses in the 10.20.0.0/24 range, which are provided by the DHCP server on the gateway. Packets from this network destined for machines on the defender network are passed through with no modification since many of these packets are likely to be exploits or other types of malicious traffic. The attacker network is shielded from incoming connections from the defender network by the defender network's filtering, but attackers are still susceptible to attacks from fellow attackers. The ideal way to isolate attackers from each other is to create a separate network for each attacker. Since we lacked the hardware resources to do so, the solution we have implemented relies on ACLs to control the network traffic of attackers.

Using a Cisco Catalyst 3550, attacker communications can be limited with VLAN-layer ACLs. The attacker network ACLs prohibit machines on the attacker network from communicating with any other host on the attacker network, except for the gateway

machine's attacker network interface. This approach is significantly simpler than segregating each attacker on their own VLAN, and is more scalable. Even with these countermeasures in place, all participants are forewarned that they are plugging into a hostile network at their own risk.

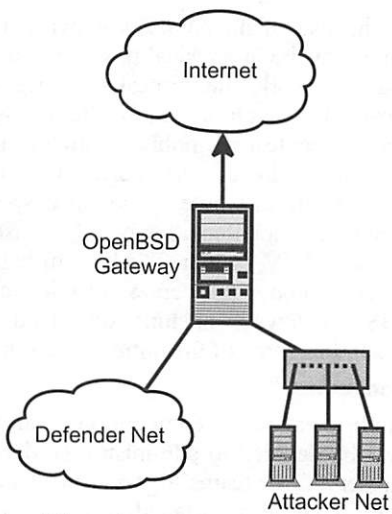


Figure 1: All routing, policy (and proxy) functions.

Gateway and Routing

Because the routing and communication policy between networks is so important to the success of the network, the policy was implemented using the method with which the administrators of the network were most familiar. Currently, the routing policy is written using OpenBSD's packet filter, pf [3], but there is work in progress to port this configuration to Cisco IOS.

The packet filtering rules were created using a set of class-based queues designed such that administrative tools like Secure Shell and Remote Desktop are allotted 20% of the available bandwidth, and given a higher priority than other traffic on the network. This was done to minimize the negative effects of network saturation on any segment of the network.

We also learned – the hard way – that the default size of the OpenBSD state table is far too small to run a contest like this. Shortly after the contest began, the state table was filled to capacity and stalled network communications. We have since increased the state table to store 20,000 states, and optimized the state timeouts to be more aggressive.

The hardware used for the gateway was a 500 MHz Pentium II with 256 MB of RAM and three network cards. Although this may not seem like a lot of computing power, the machine performed well under the strain of the contest. The primary duty of this machine is the routing between networks and performing NAT for both the attacker and defender networks. In addition, this machine also provides DNS resolution for the attacker and defender networks using DNSMasq [4] and provides DHCP leases to machines on the attacker network. In the future, this machine may also serve as a web and ftp proxy for the attacker and defender network.

This network began as a simple test network and has evolved into a secure environment for network security testing. The potentially harmful traffic is isolated to this network, while still allowing participants access to the Internet. This network provides a safe environment in which sysadmins of all skill levels may experiment without endangering any important data. This environment can be used to stage a wide variety of testing scenarios, training sessions, and security challenges.

The Game

One possible use for the above network is a game that we have created, based primarily on an annual Defcon tradition now known as Root-Fu [5, 6], formerly known as Capture the Flag. Each team of participants is assigned a host on the defender network, which they must secure against the attacks of the other teams. Teams are awarded points for each successful test of their services, and each successful compromise of an opponent's service. Teams must balance offensive and defensive tactics to gain as many points as they can over the duration of the contest. However, there are some restrictions on the tactics that may be used in this contest.

Rule Guidelines

Laying down rules for a security challenge is more difficult than it seems, and enforcing rules is even more difficult. Strict rules may prevent participants from making use of some useful attack strategies. Rules that are too lenient could result in teams using unfair methods to give themselves an advantage. The current rules do not allow for ARP spoofing or Denial of Service attacks, because these techniques have been used to disrupt the contest in previous years. The rules allow the contest administrators the ultimate authority in determining if a team's behavior constitutes "unsportsmanlike conduct," and penalizing them accordingly, either with scoring penalties or expulsion from the contest. To spot potential violations, contest administrators keep a close eye on traffic in the defender network. Contestants are also provided with a private, direct means of communication with the contest admins, in the event that they suspect their opponents of violating the rules. Fortunately, rule violations have been rare since most contestants understand the spirit in which the game is run.

The newest addition to the contest rules is the concept of machine ownership. If a team has gained administrative access on another machine on the defender network, the team may request ownership of the machine. This is done by filling out a "Change of

Ownership" report in the contest web tools, discussed later. The contest administrators then update the ownership of that machine in the database, and the team begins to earn defensive points for any services offered by that machine.

Getting Started

The first step in getting a competition like this running is recruiting participants. Each year, several members of the administrative team are tasked with creating advertising for the contest. Past advertising campaigns have consisted of posters and word of mouth, but more recently we have invested in a free-standing sawhorse sign featuring a skull and cross-bones. Once participants are interested, they will need to register.

Team registration is done using a web-based tool that stores registration information in a database, which is used later for scoring. The registration form includes all the basics like name and contact information, in addition to an Operating System preference, which allows teams to choose a preference from a list of available OS choices. This list includes at least one Unix variant, and one Windows variant. The registration system should be made available no less than one month before the scheduled beginning of the contest. This insures that interested individuals have enough time to find other interested parties to join with in the creation of a team. In order to give the administrators enough time to complete the initial setup, registration should close approximately one week before the contest begins. Once the number of registered teams is known, the real work can begin.

Initial Contest Setup

The most time-consuming portion of running this contest is the building of machines destined for the defender network. In addition to the drudgery of OS installation, members of the administrative team often sabotage the default security of the OS by adding administrative users with easily guessed passwords, or disabling common security features. In order to grant the teams access to their machines, the administrators of the contest must maintain a list of the passwords set for the Administrator/root account, so that the passwords can be distributed to the contestant team. Since this is the least interesting part of running the contest, there are efforts under way to automate this part of the procedure.

In early contests, the participants focused primarily on defense, which made for a relatively unexciting contest. In order to avoid this, a new class of machines also inhabits the Defender network for this competition. A variety of Non-Player Computers, or *NPCs*, were set up, and deliberately left vulnerable, to give participants something at which to point their attacking skills. These NPCs are built from spare hardware found in storage rooms, and frequently include devices like printers, or similar networked devices, just to keep the competition interesting. Often, the task of setting up NPCs is delegated to a newer student, without much experience in system administration. The student can then observe his NPC to see how well the configuration fares against the attacking skills of the contestants and learn from any mistakes. In this way, even students involved in the administration of the contest can gain experience.

Since the use of an Attacker network requires that the contestants be in physical proximity to the rest of the contest network, the contest now needs some physical space in which it can be held. This also requires that the contest be mobile enough to relocate for the duration of the contest. Fortunately, the CtF administrative team was able to obtain a spare 19" rack, in which they mounted a keyboard, mouse, monitor, and 16-port KVM switch. The bottom half of the rack was used to house the contestant machines, and the OpenBSD gateway machine described earlier, which also handles some of the duties of scoring.

Contest Timetable

On the first day of the competition, a fair amount of time must be devoted to administrative details and the rest is reserved for teams to work on securing the machine they have been assigned. The first administrative task is distribute information packets to each team. The administrators must verify the identity of the team leaders and the team leaders are then entrusted with the information packet. This information packet contains an official copy of the contest rules and the password for the administrative account on their assigned machine on the defender network. Also in the packet of information are instructions for connecting to the contest network from elsewhere on the Internet. With the creation of an attacker network, the remaining members of the administrative team are making sure that the machines contestants have brought with them are properly connected to the attacker network.

For the first day, participants are not allowed to attack their opponents. A member of the administrative team watches the network carefully, using Snort [7] as an Intrusion Detection System to catch any teams who violate this policy.

On the second day of the contest, the gloves come off, and the participants may begin to attack each other. For the remainder of the contest, the machines on the defender network will be the target of countless port scans and exploits. The second day of the contest also marks the beginning of the scoring period, which continues until the end of the contest.

Scoring Mechanisms

The scoring mechanism used in this contest is divided into two types: defensive and offensive. The defensive portion of the score is derived from the availability of the services offered by the team's assigned machine, as tested by an automated system. Offensive scoring has proven very difficult to automate,

and is currently handled through a web-based reporting system discussed below.

In order for an automated service checking system to be effective, it must accurately simulate user interactions with the service. If the service verification is not thorough enough, a team could replace a complex service with a simplified "stub" service. Fortunately, the Nessus remote security scanner [8] provides a flexible plugin architecture suitable for the needs of this contest. Nessus plugins can be written in either C, or the Nessus Attack Scripting Language (NASL) [9]. All contest-related service tests are written in NASL, and report success or failure using two of the built in Nessus plugin return states. The assignment of status is arbitrary, but must be consistent with the Nessus output parser.

Defensive scoring is divided into "rounds" that last ten minutes. During each round, a Nessus client connects to the Nessus daemon running on the gateway and requests a test of the defender network using only the contest-specific tests. After the server performs the checks, the data is fed back to the client as XML. The Nessus output is then parsed using XPath expressions and the result of each test is recorded in the scoring database. Storing the test results in this manner not only allows multiple tests for a single service, but if a test proves to be unreliable on the contest network, it can be disqualified without losing any additional scoring data. SQL queries can then be used to determine which services on each machine passed all tests, and should be awarded points. Point values for each service are also stored in the database, so that a team's points could be calculated using a single SQL query.

Of course, the score of the game is not nearly as exciting unless it is displayed for all to see, so some of the more graphically gifted members of the administrative team put together a scoreboard using Macromedia's Flash [10]. The scoreboard retrieves a simple XML file containing various scoring-related quantities, and displays them in interesting ways. For example, the percentage of running services is represented as a gauge, and the overall team score is displayed simply as a number. There are many other quantities that can be measured in this setting and these quantities are presented only as examples. The scoreboard is a valuable addition to the contest, as it allows contestants and spectators alike to get a clear picture of the contest standings.

For the remainder of the contest, the only task for the administrative team is to respond to feedback from participants. Teams may provide this feedback to the administrative team using a set of web tools that were created to help the contest run more smoothly.

Web-Based Administrative Tools

The CtF web tools were created to facilitate communication between the contest admins, and the contestants. As the contest became more complex, so have the tools. The tools were originally written using AxKit [11], but work is underway to remove that dependency. The registration tool is not protected by authentication, but all the remaining tools require a team username and password unless otherwise noted.

The most fundamental of the web tools used in the administration of the contest is the registration form. The registration form is filled out by the teams before the contest, and includes information such as team name, OS of choice, and the name and contact information for each member of the team. This information is recorded in the scoring database for later use by scoring software and web tools.

When a team compromises another machine on the defender network, they must fill out an offensive report. The offensive report requires that a team specify the IP address of the machine they have compromised, and provide the administrators some measure of proof that they have gained elevated access on the machine. This can be done in a number of ways, including modifying web content or binding a shell to a specific port. Any proof of compromise must be verifiable without local access to the machine, since the contest administrators do not maintain any level of access to contestant machines. The administrators then use a correspondence web tool to reply to the teams, indicating whether their proof was sufficient or not.

In the event that a team's assigned machine becomes unusable, the team may use the web tools to request that their machine be re-installed. Re-installation is done at the convenience of the contest administrators, but once rebuilt, the machine will come up unpatched and mis-configured on a hostile network. Because of this, re-install requests are rare, and may be removed from the contest in the future.

If a team gains complete control of another machine on the defender network, they may request ownership of the machine, in order to gain more defensive points for services offered by that machine. The web tool requires that the team provide the special flag, stored on a CDROM, and readable only by the administrative user. When the contest administrators receive a change of ownership request, the ownership of the machine is updated in the scoring database, and the flag on CDROM is changed to prevent the previous owners of the machine from re-claiming it. The replacement scoring system in development handles ownership at a service level, rather than a host level, so this system will be obsolete.

Teams are also encouraged to turn each other in for violations of contest rules, since the contest administrators cannot hope to catch all violations themselves. If a team believes that they have found a rule violation, they fill out a web form detailing whom they believe to be violating the rules. The contest administrators will then investigate and respond to the team who reported the violation.

If a team wishes to convey a message to the contest administrators that does not fit into one of the categories above, a general comment tool is also provided.

The administrative side of the web tool suite presents the user with a list of reports that have not yet been handled. Resolving a report emails the contest administrator's comments to the captain of the team who filed the report. These tools allow the contest administrators to efficiently handle. feedback from contestants.

This particular game has evolved to its current state over the last few years and is only an example of the many possible uses for this network. Several similar games are in development, including a "King of the Hill" type game where contestants would attempt to gain root access on a machine, and attempt to maintain that access as long as possible. Both the network setup, and the game have evolved a great deal to reach their current status, but there is much more work still being done to improve them.

Future Improvements

Over the years, this contest has improved as members of the administrative team have found ways to automate and simplify various aspects. In the spirit of continuous improvement, there is still a great deal of work in progress to make the network easier for administrators to set up and the game more enjoyable for the participants.

Network Improvements

With the creation of the attacker network, the administrators found that the network was transferring significantly more data than it had before, due to downloading and web browsing by the participants. In the past, contestants connected to the contest from their homes, so the contest was not responsible for providing normal access to the Internet. However, with the advent of the attacker network, the contest network must provide enough bandwidth for teams to research service protocols, and investigate possible attack vectors.

In order to achieve this, the routing policies described initially will be enforced by the Cisco Catalyst 3550, which performs many routing functions in hardware. Relocating this functionality will increase the throughput of the contest network, and free up the gateway for other tasks.

One task that the gateway's resources could be used for is a proxy server. Currently, the routing configuration of the gateway does not have the ability to restrict traffic to the HTTP and HTTPS protocols, merely to their respective ports. Using an application-level proxy would reduce the potential for misuse of the network by ensuring that only valid requests pass through.

Game Improvements

The most time consuming task involved in building the contest is the bulk creation of machines for use on the defender network. Although identical machines can easily be constructed using disk imaging software, a network full of identical machines is not very interesting. Research is currently under way to investigate the potential use of other configuration management systems, such as Fully Automated Installation [12], or Radmind [13] to create basic installations that are similar, but not identical. The use of such systems would drastically reduce the amount of work required for contest setup, and may allow the contest to be run more frequently.

Automation of the defensive scoring for Capture the Flag saved the administrative team quite a bit of time and effort, while automating the offensive scoring could save even more. The automated scoring system in development combines the defensive and offensive scoring into a single system based on dynamic service flags.

Each service must make the flag information available to all clients, and the flag must be located in a pre-determined location for each service. For example, the web server on host 10.10.0.3 would make its flag available at http://10.10.0.3/flag.txt. Other services must make their flags available in a similar manner.

At the start of the contest, each team is issued a special initialization flag. When a team takes control of a service, they replace the existing flag with their own initialization flag. In the next round, the scoring system will recognize that the ownership of that service has changed.

Each round, the scoring system connects to a scoring daemon on the target contestant machine, and retrieves the flag for each service from the filesystem. The scoring system then connects to the service itself and performs a series of validation tests, which includes retrieving the flag for the service. If the service flag from the filesystem does not match the flag obtained through the service, the contestant has attempted to trick the scoring system, and should be penalized. If the two flags match, they are compared to the expected value stored in the scoring system's database. If the flags match, the service is still under the control of the same team, and should receive points. If the flag retrieved by the scoring system matches the initialization key of another team, the scoring system updates the ownership of that service to reflect the compromise, but no points are awarded. This was done to deter teams from simply replacing their flags with initial flags each round.

If the flag does not match the expected value or any team's initialization key, the flag has been tampered with, and the contest admins should be alerted. As of this writing, the above system is under active development, although not yet completed.

Several contest additions have been proposed to add additional realism. The best way to add a realistic angle to the contest is to present each contestant team

with content for each service they run. Teams would be required to serve this content in order to earn points. Service tests could then be used to verify that the provided content was still in place. The number of services could also be narrowed, allowing contestants to focus their research efforts on a smaller number of protocols and allowing administrators to write more complex functionality tests for those services. Bandwidth usage and performance are important factors as well, and may be incorporated into the contest in the future.

The bandwidth used by each contestant machine could be tracked, and the teams could be "charged" some number of points, based on their bandwidth usage in each scoring round. The tracking of bandwidth usage could easily be performed by the switch, but this information must still be fed back into the scoring system. A similar "charge" could be applied for slow response to scoring checks. For example, if the response time for a particular host is significantly longer than the response of the other hosts, that host should be penalized for their poor performance. Implementing these penalties has not yet been attempted.

Although this contest has evolved for several years, there are still many more performance enhancements and features that are in development. Suggestions and feature requests are welcome.

Availability

The code and configuration files used in the contest will be made available at http://www.nerdcircus.org/ctftools/.

Conclusions

There are many reasons for an organization to run a competition like this, whether for educating new sysadmins, or just to keep current sysadmins sharp. Security training exercises like the one described here can be a great benefit to any organization. Such exercises can be used to raise awareness of common security problems and their solutions. In an academic environment, these exercises can be used to give interested students a safe environment in which they may explore many aspects of security, without endangering the security of others.

Corporate environments stand to gain just as much. Using the techniques described here would allow organizations to better evaluate the technical proficiency of the individuals being considered for security-related positions. In addition to entertainment, these training exercises could be used to keep existing members of security teams at their best.

The network described here can be built using hardware that many organizations may already have laying around in storage rooms.

Although these contests are a valuable learning experience, and have numerous other uses, the main reason that we have continued running them is simple: fun.

References

[1] Defcon, http://www.defcon.org.

[2] Rekhter, Y., B. Moskowitz, D. Karrenberg, G. J. de, and E. Lear, "Address Allocation for Private Internets," *RFC 1918*, Internet Engineering Task Force, February 1996.

[3] *OpenBSD*, http://www.openbsd.org .

[4] Simon Kelley, *Dnsmasq*, http://thekelleys.org.uk/dnsmasq/doc.html .

[5] The Ghetto Hackers, *Root Fu!*, http://ghettohackers.net/rootfu/ .

[6] divide and dd, "Root-Fu; Rise of the Ninjas," In *Toorcon 5*, http://www.toorcon.org/slides/rootfu-toorcon.ppt , 2003.

[7] Roesch, Marty, *Snort*, http://snort.org .

[8] Deraison, Renaud, *Nessus*, http://nessus.org .

[9] Deraison, Renaud, *NASL 2 Reference Manual*, http://nessus.org/doc/nasl2_reference.pdf , February 2002.

[10] Macromedia, Inc., http://www.macromedia.com .

[11] Apache Software Foundation, *AxKit*, http://axkit.org .

[12] Lange, Thomas, "FAI – Fully Automated Installation," *Eighth International Linux-Kongress Proceedings*, November 2001.

[13] Craig, Wesley D. and Patrick M. McNeal, "Radmind: The Integration of Filesystem Integrity Checking With Filesystem Management," *LISA '03 – Large Installation System Administration Conference*, October 2003.

Securing the PlanetLab Distributed Testbed

How to manage security in an environment with no firewalls, with all users having root, and no direct physical control of any system

Paul Brett, Mic Bowman, Jeff Sedayao, Robert Adams, Rob Knauerhase, and Aaron Klingaman
– Intel Corporation

ABSTRACT

PlanetLab is a globally distributed network of hosts designed to support the deployment and evaluation of planetary scale applications. Support for planetary applications development poses several security challenges to the team maintaining PlanetLab. The planetary nature of PlanetLab mandates nodes distributed across the globe, far from the physical control of the team. The application development requirements force every user to have access to the equivalent of root on each machine, and use of firewalls is discouraged. If an account is compromised, PlanetLab administrators needed a way to track the actions of users on the nodes. If an entire node is compromised, then the administrators need a way to regain control despite the lack of physical access. Encryption was built into PlanetLab to ensure confidentiality and integrity of system downloads. A special reset packet, combined with keeping a boot CD in the machine, enables PlanetLab system administrators to remotely regain control of machines if they are compromised and return to the nodes into a safe known state. The Linux VServer implementation is used to provide root access to PlanetLab users for development purposes while isolating users from each other. A network abstraction layer provides accounting of traffic and allows safe access to raw sockets. These mechanisms have proven very useful in managing PlanetLab. After a compromise of large numbers of PlanetLab hosts, control of the PlanetLab network was regained in 10 minutes. The compromise spawned a review of PlanetLab security, which pointed out a number of flaws. The need for a central site for maintaining PlanetLab was cited as a key weakness. Future work includes distributing the functions of PlanetLab's central administrative database and improving integrity checks.

Introduction

The PlanetLab distributed system testbed [1] has a number of unique attributes that make security administration difficult. PlanetLab is a globally distributed network of hosts designed to support the deployment and evaluation of planetary scale applications by distributed systems researchers all over the world. To support their application development efforts, PlanetLab users need root access. PlanetLab systems are used for a variety of network experiments and require unfettered access to and from the Internet. Use of firewalls between PlanetLab hosts and the Internet is strongly discouraged. Moreover, PlanetLab systems are at sites distributed all over the planet and are not under the direct physical control of any of the PlanetLab administrators. At the same time, PlanetLab nodes must be available and usable by the researchers while site and PlanetLab administrators need to be able to respond to security complaints.

This paper describes the security challenges faced by the PlanetLab administration team. It reviews the issues that needed attention, how we dealt with them, and our experiences with the implementation.. We first describe the PlanetLab environment and the system and network requirements of the PlanetLab community. We then describe our design and the implementation of that design. After that, we review our experiences with this design, followed by sections on related and future work.

Environment and Problem Description

In this section, we describe the PlanetLab environment and talk about the key security challenges facing PlanetLab administrators.

PlanetLab Environment

PlanetLab [1] is a distributed-systems testbed, allowing the research, development, and prototyping of new applications and network services. The testbed comprises 433 nodes at 194 sites.[1] PlanetLab is truly "planetary scale" as it is geographically spread across five continents and topologically spread across the Internet, Internet2, and other networks. Because of this geographic and network diversity, the test bed provides researchers with a very "real-world" set of opportunities and challenges; specifically it allows the deployment of, experimentation with, and test/measurement of services in a non-simulated network.

PlanetLab nodes are computers (PCs that meet an evolving set of minimum configuration requirements)

[1]As of the time of this writing; growth continues.

which are hosted by a variety of universities, corporations, and other organizations. The nodes are dedicated running PlanetLab. They run a special version of Linux (detailed later in the paper) and are administered remotely (including patching) by a set of administrators focus on PlanetLab. The set of initial test bed machines was seeded by a grant from Intel Corporation, but now new member organizations must contribute nodes as a condition of joining the test bed. Hosting organizations provide electrical power, physical space, and network connectivity to their nodes. PlanetLab administrators have physical access to almost none of the nodes, and the turnaround on physical work requests on the machines is on the order of days. Administrative and system burden on the hosting organization is deliberately limited (we don't require remote consoles, for example), in order to make joining the testbed as painless as possible.

Security Challenges

Because of the nature of the research done on PlanetLab – requiring unfettered access to the network and frequently resulting in non-standard traffic patterns – the nodes are generally positioned outside of the hosting organization's firewall. A consequence of this is that nodes are typically not protected by any kind of filtering of inbound traffic, and lack of the outbound filtering permits all kinds of traffic, some of which will be interpreted as hostile, to be sent out.

PlanetLab users perform distributed systems research. Accomplishing this frequently requires great flexibility on the part of the system – for example, requiring root access to perform certain functions, or wishing to use the system in odd ways (or replace part of the system with their own code). At the same time, the node must remain stable enough for use. Moreover, researchers don't want other experiments affecting their experiment (as well as the converse).

PlanetLab nodes are administratively very complex – they consist of machines in different networks and administrative domains, providing access to researchers who are at arbitrary locations in other administrative domains and who do odd, non-standard things. Keeping control of the nodes is a difficult problem. That control rests with a centralized group of PlanetLab system administrators who develop and maintain the base operating system (including patches for security and for PlanetLab functionality) and the associated management utilities.

Host organizations also frequently have requirements that they be able to control nodes on their network. One concern that host organizations have is that PlanetLab nodes would be used for nefarious purposes. Sites would need ways to audit PlanetLab usage to help them deal with any possible complaints that received about PlanetLab node behavior. Despite the trust that sites have shown by hosting PlanetLab nodes, we anticipated that at some point PlanetLab nodes could be compromised. We needed a mechanism be able to remotely regain control of hosts when this happened. Nodes would need to be brought to a safe known state for forensics and for removal of vulnerabilities. Also, the ability to remotely power cycle a node would not be enough. While that functionality is extremely useful for remote managing machines when they get hung, it does not implicitly put the PlanetLab node into a state where it can be debugged remotely.

[1]The PlanetLab consortium is hosted by Princeton University, and their staff serve as centralized PlanetLab system administrators.

Summary

The security key problems faced are summarized as the following:

- Create a full and rich development environment where users have tremendous flexibility while being isolated from each other and the native OS environment.
- Make it possible, even comfortable for sites to host PlanetLab nodes despite possible complaints about node behavior and the fact that the local site does not fully control the node
- Be able to regain control of PlanetLab nodes even if they are compromised.

Related Work

There are a number of large distributed/network testbeds that deal with similar issues. Emulab [2] is a network testbed that has many of the same concerns as PlanetLab. Emulab uses the FreeBSD jail [3] to isolate experiments in a type of virtual machine. PlanetLab differs from Emulab in that PlanetLab emphasizes the development of services and APIs and also aims to be a deployment platform for services. Also, while Emulab nodes talk primarily to each other, PlanetLab nodes are encouraged to and often do communicate with non-PlanetLab nodes, making the need for an audit trail of PlanetLab node more critical.

As we describe later, PlanetLab uses a virtualization technology to isolate users from each others while giving them a very flexible environment, not unlike Emulab's use of a chroot jail. Related work in the virtualization area are Xen [4] and Denali [5]. A number of PlanetLab nodes are even running on top of Xen.

Other work has been done to manage an environment where users have tremendous flexibility and need the equivalent of root. Leon, et al., [6] discuss how they manage an environment where all users have root. Like the environment described, PlanetLab takes advantage of having sophisticated users who are willing and capable of managing their own environment. PlanetLab is not intended as a desktop environment where users perform activities such as receive mail.

The Grid has been compared to PlanetLab in [7]. PlanetLab differs in that it is more network centric vs. compute-centric than the Grid. Many PlanetLab applications, such as network measurement [8] and content

distribution [9, 10], rely on geographic and network dispersal to be effective, while rarely being CPU intensive. Dispersal of Grid resources tend to be more accidental than intentional. Also, Grid resources are shared (compared to dedicated PlanetLab node) and typically more heterogeneous than PlanetLab nodes and are not centrally managed like PlanetLab. The network-centric application mix of PlanetLab, particularly with network measurement and content distribution, makes the audit trail requirement more pressing.

Security Design and Implementation

There are many points of control that must be managed to provide top-to-bottom control of the system. The first are the users who must use the system's resources in a reasonable way, as not to provoke sites into removing the hosts. Next, resource utilization, such as network, CPU, and disk space, for each user must operate within certain bounds. The platform must be remotely controlled. The booted operating system and base execution environment must be installed securely and controlled remotely. This section describes each of these pieces and how they are controlled and secured.

AUP

The PlanetLab users operate within the limits of a published Acceptable Use Policy (AUP). All PlanetLab users get access to PlanetLab by first creating an account at PlanetLab Central. One step in this signup is the user's acceptance of the AUP. Since PlanetLab is a testbed for experiments in new Internet technologies, it is difficult to enumerate specific limits within which users must operate. Of course, malicious activity, attempts to subvert the PlanetLab security and authorization system, illegal activities, excessive node use and activities that exceed the usual limits of network propriety are called out, but the general rule is "do no harm." The AUP instructs PlanetLab users to ask what activities would cause network and resource alerts in their own site and then consider the same sort of limits on the remainder of the PlanetLab modes.

User Isolation

To provide a rich development environment to users yet provide user isolation, we modified the base OS of PlanetLab nodes [11]. PlanetLab administrators use a lightweight virtual machine abstraction provided by the Linux VServer [12] implementation. Each research group getting access to a node receives a chrooted virtual Linux machine, which we will call a vserver. The user API effectively becomes Linux. VServers virtualize machines at the system call level, above the kernel. Virtualizing at this level allows us to scale to 1000 virtual machines at the cost of weaker isolation, something not possible with other VMM implementations like VMWARE [13] or Xen [4]. To use the PlanetLab network, researchers get "slices" of the infrastructure. Slices are collections of accounts on some set of nodes across the network. These accounts on a node are isolated within vservers, with the exception of some administrative slices.

Network isolation is achieved through a "safe raw sockets" implementation [14], part of the SILK package, which is derived from Scout [15]. This implementation provides controlled access to the network stack by what appears to be raw sockets without granting root privilege. It also isolates traffic, preventing individual virtual machines from snooping on each other's traffic. In addition to isolating network traffic, SILK provides CPU guarantees and enforces usage policy. The Linux Traffic control facility [12] is used to manage the bandwidth utilization and implement bandwidth policies. We allow site administrators to set the amount of bandwidth that each PlanetLab node can use.

SILK also provides network traffic auditing capability. SILK tags each packet with the ID of the VServer that sent it and provides an administrative port for snooping outgoing traffic. We also created blacklists that would prevent a PlanetLab node from contacting some set of IP addresses. PlanetLab administrators install these blacklists, and local site administrator can request that nodes or networks be placed on them. Care needs to be taken in the installation of blacklists to prevent nodes from being made totally inaccessible.

Reporting

PlanetLab's geographic distribution makes it ideal for mapping the Internet. It seems that many researchers first build a "Hello world" application that pings other PlanetLab and non-PlanetLab nodes to discover timing and connectivity information. Repeated pings, IP address space scanning and port scanning are just the activities that set off Snort [16], and other network monitoring tool alarms. Even some "well designed" probing applications (i.e., with built in flow restriction to avoid complaints) have set off alarms. This implies that some sites have very tight restrictions on probing and mapping activities.

To handle an inappropriate traffic incident, we need to map the reported activity from the network traffic to the experimenter. A traffic report usually contains a time and a source and destination IP address. Additionally, traffic reports relate to an incident in the past. We found that most conventional traffic monitoring tools are for watching current traffic and not recording and querying past traffic.

These problems (mapping, delay and distribution) led to the development of tools which have each node collecting information on its own network traffic (in and out), saving that information and eventually reporting that information to a central repository.

As mentioned above, the kernel's network stack was enhanced with SILK to return information on which IP addresses to which the slivers were communicating. An administrative application named "netflow"

analyzes this information every five minutes and calculates the "flows" – the connections from a source to a destination by some slice. This information is saved to a file. These files are kept on the node and are eventually copied to PlanetLab Central where they are available for analysis if problem reports arrive.

This flow information is then made available on each node and from PlanetLab Central. Each PlanetLab node runs a web server on the standard web port (80) that gives information about PlanetLab, about the node and allows browsing through the flow information. This allows local administrators of sites hosting PlanetLab nodes to respond to traffic and security alerts. Through the web page, an administrator can search for the reported destination IP addresses and trace this to the email address of the researcher. If a remote (not at the PlanetLab site) network administrator receives reports about the traffic from a PlanetLab node, that administrator can contact the experimenter directly. In this way, the reporting facility removes PlanetLab administrators from the chain of contacts regarding a perceived incident, reducing the time to respond to security complaints. Given the exposure of this web server (no passwords or accounts are needed to access it), web pages are implemented in simple HTML (no java, javascript, or PHP) with no user text input required for selecting looking at traffic patterns.

The requirement on the PlanetLab infrastructure for providing this service is maintaining a mapping between service operators (the researchers). This means a verifiable system of user identities and the monitoring system that records the user of resources and who is using them. PlanetLab thus has a extensive system of monitoring resource use and this monitoring system is tied into a system that authenticates users and which provides a path back to the email of a responsible person for any resource use.

Preventing and Dealing with Compromises

We knew that there was a substantial chance that our exposed network of nodes could be compromised. To deal with that possibility, we configured our machines to boot only from a CD in a machine. Once the machine boots, it downloads via an SSL secured connection a gpg signed script to execute as the next phase of the boot process. These scripts are used for remote re-installation, normal booting, and placing the node into a "debug" mode in which the network stops all traffic except ssh connections. Since scripts for normal reboot are downloaded from a central location, we can upgrade the kernel versions used on the hosts without having to update the CD. During debug mode, should the connection to the PlanetLab central website become unavailable for any reason, the node will reboot and retry the connection at 15 minute intervals. With the debug mode, we can bring nodes into a safe known state while preserving disk information for forensics.

The Linux kernel on PlanetLab nodes has been modified to reboot when it receives an ICMP trigger packet with a unique 128 bit payload which is generated for each machine and is re-generated each time a machine reboots. We choose 128 bits to make exhaustive search attacks very difficult against a single node. Since each node has a unique packet for reboot, replay attack is ineffective against the nodeentire PlanetLab network. At worse, a replay attack will only cause a single node to be rebooted. If desired, the machine can be forced to come up into a special debug mode, to which as described above, limits access while allowing for forensics. While effective, this software reboot mechanism suffers from the problem that the machine must have a working network stack, and connectivity to the internet in order to ensure a reboot. With the widescale filtering of ICMP traffic following the SQL Slammer worm, we now recommend that PlanetLab sites install remote power switches on their nodes.

Experiences

Many of the security features implemented in have proven very useful. Our reporting mechanisms have defused many incidents after a network experiment triggered an overly sensitive Intrusion Detection System (IDS). Remote control and access made recovery from a system compromise quick and effective. This section will describe some of the incidents and successes of the PlanetLab security and control mechanisms.

User Behavior and AUP

Since the current direct users of PlanetLab are researchers who generally understand operation on the Internet, there have not been many incidents that required enforcement of the Acceptable Use Policy ("AUP"). We have not yet had to revoke any access from user, which would be the ultimate penalty associated with AUP violation.

Problems with PlanetLab user behavior have been studied [17] and fall into two categories: program failure and accidental network traffic alerts. Building distributed, decentralized applications is hard and, of course when you have lots of projects building them, there will be bugs. PlanetLab Central will receive reports or will notice excessive node resource use (e.g., no file descriptors) or excessive network traffic (e.g., too many external computers accessed or excess volume) and PlanetLab Central sends email to the researcher. In all cases, the researchers have responded to the situation.

Measuring the Internet generates lots of probes and pings. A simple mapping experiment, generating a small amount of data and performing a straightforward measurement set off alarms at many locations. In this case, and in others, a measurement experiment has the same network traffic profile as a worm looking for hosts to infect (probing port 80 is a feature of CodeRed/NIMDA). It is against the PlanetLab Acceptable User Policy to generate "disruptive" network traffic but it's sometimes hard to know what type of traffic would be considered disruptive.

There have been many incidents of "attack" or "worm" reports that were traced back to a measuring or topology experiment. The resource monitoring system allows forwarding of the reports to the researchers and, in all cases, the researchers responded appropriately to the situation.

Monitoring

We put a lot of energy into providing ways for network and security administrators to determine for themselves what researcher generated traffic and how to contact them. While these facilities did reduce the workload from dealing with security complaints about PlanetLab node behavior, we continue to have problems from overzealous intrusion detection systems. Some organizations set up intrusion detection systems (IDS) to trigger on relatively innocuous things as a traceroute. One organization went as far as to threaten lawsuits if behavior persisted, and this tactic proved successful in getting a number of PlanetLab hosts pulled off networks. Sometimes complaints were justified, as some researchers experiments generated what would have to be interpreted as an attack – large numbers of connections attempted to a range of IP addresses in a domain. In any case, we anticipate that poor experiment behavior and overly sensitive IDS will continue to cause problems.

Compromise and Recovery

We had anticipated that at some point, PlanetLab nodes would be compromised, and we did have an incident where large numbers of PlanetLab nodes were compromised. The early implementation of PlanetLab had accounts that were not virtualized – they had access to the native operating system. An SSH key to a non-virtualized account was compromised, and that key was used to log into a number of nodes. Since the account was not isolated within a VServer, the attacker used his access to the native operating system obtain root. When we received notice that a number of PlanetLab hosts had been rooted, we used the reboot feature of PlanetLab nodes to force all of PlanetLab into a known safe state in 10 minutes.

We took a number of actions in response to the compromise. Forensic analysis, enabled by debug mode, determined that the nodes were rooted using a vulnerability that we had plans to patch. We had just begun to roll out a version that was not vulnerable to the exploit when we were attacked. We also eliminated general purpose slices that were not isolated with VServer accounts. At the same time, we made all user slices dynamic and eliminated static VServer slices. Slices would not be assigned by default to all nodes, which would limit the access to nodes if a slice private key were compromised. In addition, slices would have a finite lifetime. They would not last indefinitely, and would need to be renewed. This idea brings us closer to the idea of least privilege for slices – slices would only be instantiated on nodes that they needed and only for as long as they were needed.

A Security Review

We had fixed the more obvious problems, but what about problems we had not yet anticipated? Another response to the incident was to have more eyes looking at the problem, so we conducted a review of PlanetLab security. We wrote a summary of PlanetLab's architecture and implementation and had it reviewed by a variety of security researchers and practitioners and PlanetLab users. This review proved quite valuable, finding a variety of vulnerabilities and areas of improvement at both the implementation and architecture levels.

One key problem found was the dependency on a single instance of PlanetLab Central for key PlanetLab's operations, such as slice creation and deletion and software updates. A compromise there would lead to a compromise of all PlanetLab nodes. A DOS attack on PlanetLab Central, while not rendering PlanetLab unavailable, would make many key PlanetLab functions unusable.

Another problem regarded resource management. We need better ways to monitor and manage resources such as slices, CPU, disk, and bandwidth utilization. A runaway process could render nodes unusable and take up so many resources that it would be nearly impossible to log in and fix the problem. In addition, much of resource management and the security associated with it is hard to use. Slices can only be created the principal investigator at site. As a site PI this is usually a busy professor, this leads to a tendency of the PI to share his password, and we have evidence of this. Also, the dynamic slice mechanism provides no warning when slices will expire and all of our user's work will disappear. As a result, there is a perverse incentive to create slices that live as long as possible. Some of PlanetLab users had a contest to see who create the longest lived slice. Many of the reviewers mentioned that security that is hard to use will usually be worked around, as demonstrated.

Related to the resource management issues is the need for better intrusion detection and prevention. While we have worked to improve the isolation of slices from each other and then real operating system, if a PlanetLab slice is compromised, the attacker has a large amount of resources available to him. We need ways to detect resource misuse and intrusion. Also, we need better ways to authenticate and authorize users. Relying on a single database of information run by a single organization to authenticate and authorize users is not likely to scale or be secure. As the number of users and organizations using PlanetLab increases, it is unlikely that PlanetLab administrators could revoke them when those users leave an organization. Instead, a federated scheme, where access is granted to some institutions and those individual institutions manage who has valid access, is more likely to be successful in the long term. PlanetLab does not allow easy ways for slices to authenticate and authorize each other. As a result, some

users have poor practices such as leaving SSH private keys on nodes. One reviewer pointed out the use of any reusable password is not vulnerable to attack. The attacker who compromised PlanetLab set up sniffers that listened not to network interfaces but to TTY ports. In this case, SSH does not help as data streams are decrypted when the attackers are listening to them, so if users actually did put pass phrases on their SSH private keys, those pass phrases could be compromised.

Future Work

A large focus of future work is on the dependencies on a single centralized facility for much of PlanetLab's operations. Key dependencies are in PlanetLab Central are being analyzed, and a more formalized threat assessment matrix will be created. Creation of a Red team for more formally and thoroughly analyzing security weaknesses is also being considered. Integrity checkers such as chkrootkit [18] and rkdet [19] and other IDS like features are being evaluated and tested. Longer term, the architecture for PlanetLab management is being studied to make it more secure and scalable.

Conclusion

PlanetLab's security mechanisms have worked relatively well so far. The VServer mechanism effectively gives PlanetLab users a whole virtual machine to use and configure while isolating them from each other and the native operating system. The PlanetLab user account system allows network and security administrators a way to determine the source of problematic traffic. While PlanetLab has hosted hundreds of projects and researchers, and a major compromise was dealt with swiftly and effectively using PlanetLab's reboot mechanisms. A review of PlanetLab's architecture and implementation has yielded a number of areas of improvement, such as the vulnerability of having a single point of control, the need for better resource management, and the need for improvements in authentication and authorization.

Author Information

Jeff Sedayao is a staff engineer in the Planetary Services Strategic Research Project and in Intel's IT Research Group. He focuses on applying PlanetLab and PlanetLab developed technologies to enterprise IT problems. Sedayao joined Intel in 1986, where he designed and implemented almost all aspects of Intel's Internet connectivity, including routing, firewalls, mail, proxying, and DNS. After leaving Intel's IT organization, he worked in Intel's Online Services venture, designing firewall configurations, managing network services, and providing consulting services on security, mail, and DNS. Sedayao has participated in IETF working groups, published papers on policy, network measurement, network and system administration, and authored the O'Reilly and Associates book, *Cisco IOS Access Lists*. He has recently written PlanetLab Design notes on port usage and IP address usage.

Mic Bowman is a senior researcher within Intel's Virtualization Platform Lab. He received a Ph.D. in Computer Science from the University of Arizona. Bowman joined Intel's Personal Information Management group in 1999. While at Intel, he has developed personal information retrieval applications, context-based communication systems, and middleware services for mobile applications. He is currently a Principal Investigator for Intel's Planetary Services Strategic Research Project. Prior to joining Intel, Bowman worked at Transarc Corporation, where he led research teams that developed distributed search services for the Web, distributed file systems, and naming systems.

Paul Brett joined Intel in 2000 as part of Intel's Online Services group. He is currently focused on PlanetLab, a global test bed for developing, deploying and accessing planetary-scale services. From 1988 to 2000, Brett worked on the design and implementation of dependable systems for air traffic control. He is a graduate of the UKs Open University, where he earned a First Class Honours degree in systems engineering of software- based systems.

Rob Knauerhase is a staff research engineer in the Planetary Services Strategic Research Project. He joined Intel in 1993 and has been involved in the research and development of mobile networking, handheld/mobile computing, distributed computing, Internet technologies and middleware, and static and runtime compiler environments. He holds 11 patents, with approximately 60 more pending. Rob is a Senior Member of the IEEE and IEEE Computer Society and has been an adjunct professor at Portland State University.

Aaron Klingaman is a member of the research staff at Princeton University. He joined Intel in 2001 after receiving a B.S. degree in Software Engineering, with honors, from the Oregon Institute of Technology. His initial research focused on interactive television. His current interests include management of remote distributed computing and network resources. Extracurricular projects include work in the areas of firmware development and fractals. Klingaman currently works on supporting and developing PlanetLab's infrastructure.

Robert Adams began working with computers long before he earned a B.S. in Computer Science at Oregon State University. After receiving his degree, he spent ten years in Silicon Valley, where he worked on multi-processor operating systems for large and small computer systems. Adams joined Intel in 1986, initially writing drivers for the first port of Unix to Intel's 386 processor. Later, he moved to Intel's new Multimedia Systems Technology Group, where he was a principal developer of video and audio conferencing systems, eventually winning Intel's Achievement Award for leading the team that wrote the company's first software video codecs. He also has worked in Intel's Architecture Laboratory developing multiple

technologies, including cable modem networking, statistical text analysis, peer-to-peer networking and Web infrastructure. Adams holds more than 18 patents and has served on various Internet standards committees. His current work focuses on the deployment of PlanetLab, a global test bed for developing, deploying and accessing planetary-scale services.

References

[1] Peterson, L., T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," *Proceedings of HotNets I*, October 2002.

[2] White, Brian, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, "An Integrated Experimental Environment for Distributed Systems and Network," *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[3] Kamp, P. H., and R. N. M. Watson, "Jails: Confining the Omnipotent root." *Proceedings of the Second International SANME Conference*, May 2000.

[4] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Hegebar, Ian Pratt, and and Warfield, "Zen and the Art of Virtualization," *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[5] Whitaker, Andrew, Marianne Shaw, and Steven D. Gribble, "Denali: A Scalable Isolation Kernel," *Proceedings of the Tenth ACM SIGOPS European Workshop*, September 2002.

[6] De Leon, Laura, Mike Rodriguez, and Brent Thompson, "Our Users Have Root!" *Procedings of the Seventh Large Installation System Administration Conference (LISA '93)*, November 2003.

[7] Ripeanu, Matei, Mic Bowman, Jeffrey Chase, Ian Foster, and Milan Milenkovic, "Comparing Globus and PlanetLab Resource Management Solutions," PDN-04-018, February 2004.

[8] Spring, N., D. Wetherall, and T. Anderson, "Scriptroute: A Public Internet Measurement Facility," *USENIX Symposium on Internet Technologies and Systems*, 2003.

[9] Wang, L., K. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and Security in the CoDeen Content Distribution Network," *Proceedings of the USENIX 2004 Annual Technical Conference*, June 2004.

[10] Freedman, M., E. Freundenthal, and David Mazieres, "Democratizing Content Publication with Coral," *Proceedings of NSDI '04: First Symposium on Networked Systems Design and Implementation*, March 2004.

[11] Bavier, Andy, Mic Bowman, Brent Chun, Scott Karlin, Steve Muir, Larry Petersen, Timothy Roscoe, Tammo Spalink, Make Wawrzoniak, "Operation System Support for Planetary-Scale Network Service," *Proceedings of NSDI '04: First Symposium on Networked Systems Design and Implementation*, March 2004.

[12] *Linux VServers*, http://www.linux-VServer.org .

[13] *VMWARE*, http://www.vmware.com/ .

[14] *Safe Raw Sockets*, http://www.planet-lab.org/raw_sockets .

[15] Mosberger, D. and L. Peterson, "Making Paths Explicit in the Scout Operating System," *Proceedings of the Second OSDI Conference*, Oct 1996.

[16] Roesch, Martin, "Snort – Lightweight Intrusion Detection for Networks," *Procedings of the Thirteenth System Administration Coference (LISA '99)*, November 1999.

[17] Adams, Robert, "Distributed System Management: PlanetLab Incidents and Management Tools," PDN-03-015, November, 2003.

[18] Murilo, N. and K. Steding-Jessen, *Chkrootkit*, http://www.chkrootkit.org/ .

[19] *rkdet*, http://vancouver-webpages.com/rkdet/ .

Secure Automation: Achieving Least Privilege with SSH, Sudo and Setuid

Robert A. Napier – Cisco Systems

ABSTRACT

Automation tools commonly require some level of escalated privilege in order to perform their functions, often including escalated privileges on remote machines. To achieve this, developers may choose to provide their tools with wide-ranging privileges on many machines rather than providing just the privileges required. For example, tools may be made setuid root, granting them full root privileges for their entire run. Administrators may also be tempted to create unrestricted, null-password, root-access SSH keys for their tools, creating trust relationships that can be abused by attackers. Most of all, with the complexity of today's environments, it becomes harder for administrators to understand the far-reaching security implications of the privileges they grant their tools.

In this paper we will discuss the principle of least privilege and its importance to the overall security of an environment. We will cover simple attacks against SSH, sudo and setuid and how to reduce the need for root-setuid using other techniques such as non-root setuid, setgid scripts and directories, sudo and sticky bits. We will demonstrate how to properly limit sudo access both for administrators and tools. Finally we will introduce several SSH techniques to greatly limit the risk of abuse including non-root keys, command keys and other key restrictions.

Introduction

Since its introduction in 1995 by Tatu Ylonen, SSH has quickly spread as a secure way to login and run commands on remote hosts. Replacing the previous r-commands (rsh, rexec, rlogin), SSH provides much needed encryption and strong authentication features. Relying on public/private key techniques, SSH is very resistant to man-in-the-middle, IP spoofing and traffic sniffing attacks, all of which were significant problems with the r-commands. SSH was initially released under a free license, but has since split into commercial[1] and free versions. In this paper we will focus on the most popular free version, OpenSSH.

Sudo was developed in1980 to allow users to execute commands as root without using the root password. Today it provides per-host and per-command access control features and powerful logging facilities to track what is done by whom.

Setuid (also called "suid" or "Set UID") allows a UNIX program to run as a particular user. If the executable is owned by root for example, the program will run as the root user, giving it privileges that may be needed for its function. The passwd password-changing program is a good example of this, since it requires root privileges to write to /etc/shadow which holds user passwords. Setgid provides the same functionality for UNIX groups, giving the program access to files writable only by a particular group. For example, in FreeBSD programs that read system memory are setgid to a special kmem group.

These tools and features are available for all modern versions of UNIX, and are installed by default on most of them. All of them can be used to help enhance the principle of least privilege, which we will discuss at length here.

The Situation

Consider the following system administration environment and compare it with your own experience:

- SSH has universally replaced rsh, but null-password keys have been deployed to provide unrestricted root access to automation tools;
- Sudo has replaced the root password for most administration functions, but admins generally only use it to obtain root shells and almost never employ it in automation tools;
- Custom setuid scripts almost exclusively run as root and setgid is seldom used;
- Automation tools that require any root access, no matter how little, run as unrestricted root through root cron, root ssh and similar mechanisms;
- Automation tools receive little security review, even when granted wide-ranging privileges.

Such environments have been the norm in the author's experience. If you have a similar environment, this paper will introduce the ideas behind least privilege and how these tools can be used to enhance least privilege in your environment.

[1] The commercial version of SSH is owned by SSH Communications Security. The parts of this paper which refer to "commercial SSH" are based on SSH Secure Shell 3.2. Starting with version 4.0, this product is known as SSH Tectia.

The Risks

Some of the risks in the environment described above include:

- Null-password root SSH keys.[2] If an attacker can get to that key, she will have complete control over all machines that accept it. Even if your application is secure, any mechanism that an attacker can use to get to that file is fair game.
- Sudo passwords. Every account that has unrestricted root sudo access is another root-equivalent password for an attacker to guess or steal.
- Sudo hijacking. In sudo's default configuration, an attacker who can run commands as a sudo-enabled user can hijack that user's sudo privileges even without access to the user's password.
- Sudo escalation. It can be extremely challenging to limit sudo access to a few commands. Without great care, limited sudo can be trivially translated into full sudo access. While you may trust the user you granted access to, do you also trust the attacker who has stolen his identity?
- Script exploitation. Scripts that run as privileged users are obvious targets for attackers. Errors in the scripts are subject to exploitation. Setuid scripts are particularly susceptible because they are often written in scripting languages like Perl or Bourne Shell and can be read by an attacker searching for vulnerabilities.

We'll discuss how to mitigate all of these.

The Causes

It's tempting to simply blame "coder laziness" for this situation, but this isn't the case. There are several factors that we will need to address:

- Trust in "instant security." Neither SSH nor sudo can be simply "dropped in place" and deliver an ideal security environment. While SSH is far better out of the box than rsh, it has its own security issues that have to be considered, and converting automation tools to use it can be difficult without tearing down some of its benefit. Similarly, sudo introduces several security concerns, some of which are worse than what it replaces (such as a greater number of root-equivalent username/password combinations). This is not to discourage the use of these tools, but they do not magically instill security on their own.
- Lack of best practices guides. There are limited resources available explaining the best way to set up SSH and sudo. Out of the box, sudo does not even have all of its security features turned on and is subject to hijacking (as we'll discuss below). SSH command keys are mentioned in the man pages, but there are few resources really explaining their use or the use of other SSH key restrictions.
- Added complexity. Many of the techniques in this paper increase the complexity of developing and deploying automation scripts. Automation is hard enough to just get working, let alone get working securely. If developers are only rewarded for functionality, then there is little incentive to take on the added support headaches of a more secure solution.

This paper will address the first two causes. Addressing the third is often a cultural and infrastructure challenge that can only be solved on a case-by-case basis.

The Goal: Least Privilege

Now that we've discussed what may be wrong with our environment, what do we want our environment to look like? In this paper we will mostly focus on least privilege, which is one piece of the bigger goal of layered security.

Layered security means that safeguards overlap such that if one fails, an attacker will still not have damaging access. Least privilege helps ensure that if a particular user's account is compromised, for whatever reason, the damage the attacker can do with it is limited as much as possible. This is why "don't you trust me?" should never be the argument for excessive privileges. Wherever possible, trust should be compartmentalized.

UNIX-like systems provide numerous ways to restrict privileged access. In this paper we will discuss the following techniques:

- Restricting SSH connections in what they can execute and where they can originate;
- Limiting privileged access through sudo by coupling it with non-root setuid;
- Replacing root-setuid with non-root setuid and setgid;
- Reducing the number of privileged processes with sticky bits and setgid directories.

Whenever a process or user needs elevated privileges, it should be second nature to ask precisely what privileges the process or user needs, and how to best limit the process or user to exactly those privileges.

When discussing the principle of least privilege, one might ask "why would we have hired these people if we didn't trust them?" Least privilege has little to do with the trust we have for our employees. Instead, it deals much more with the number of avenues an attacker has for exploiting the system. Of course an administrator should have every access she needs, but conversely she should have no access that she has no need for. How strictly "need" is defined is a serious trade-off to consider, but just requiring that an administrator explicitly request specific access, even if it is always granted, can go a long way towards controlling the number of avenues an attacker can use. If an attacker is successful, being able to enumerate the accounts with access is also a major benefit to investigators in determining possible further compromises.

[2] Throughout this paper, the term "SSH key" will be used to refer to both RSA and DSA keys.

When granting privileges to automation tools, one might assume that the security of a particular tool isn't very important if the data it deals with is non-sensitive. It is critical to always consider how a tool could be exploited to attack other parts of the network, not just the parts it's intended to control.

Moving towards least privilege, especially for automation tools, has other benefits. Establishing least privilege requires developers to understand the privileges actually used by their tools, which in turn forces them to understand what their tools are doing. Understanding software is a key step towards maintaining it. Moreover, simply enumerating the privileges that a tool requires can help a developer see how to reduce the number of privileges required. Does the tool really need the ability to "run an arbitrary command on any host in the system" or did it really just need the ability to "get a directory listing for a specific directory on three hosts?"

Least privilege is a philosophy, not a technology. By consistently employing it, an organization can better understand and control the security of the environment while still maintaining a strong culture of trust for the administrators.

Hardening the Environment

This paper focuses on automation techniques, but some basic environment hardening will set the stage for a secure automation environment.

Understanding the Environment

In a complex environment with many users and administrators, it is easy for trust relationships to grow throughout the system with little documentation or understanding. To combat this, it is helpful to create a directed trust graph of your network, indicating particularly how root can move through the system using SSH, rsh and other mechanisms (such as custom administration daemons and web scripts that are sometimes developed in large environments). There are few tools to automate this today, but even manually developing such a graph with tools like Microsoft's Visio or AT&T's Graphviz can provide significant insight into your environment.

Understanding what users and hosts are trusted with wide-ranging root access provides a road-map for improving enforcement of least privilege. There will always be a few places in any large system that require broad trust; understanding these will give a roadmap for hardening.

Similarly, administrators should maintain a catalog of known setuid and setgid programs and audit systems regularly for the creation of new ones.

Hardening and Managing SSH

Authorized Keys

By default, SSH relies on files in the user's home directory for certain authentication options. Chief among these is the authorized_keys[3] file. This file defines what keys will be accepted without a password and under what conditions, and will be the subject of several SSH techniques in this paper. Anyone who can write to this file for a particular user can log in as that user. This means that user home directories, particularly the ~/.ssh directory, are highly sensitive. Unfortunately if home directories are NFS mounted, there are a number of ways that attackers may be able to write to arbitrary user directories, and thereby update authorized_keys with keys the attacker controls.[4] The solution is to move authorized_keys out of the users' NFS-mounted home directories and onto local storage, generally under /var. For example, the following setting in sshd_config will read authorized_keys from /var/ssh/*user*/authorized_keys:

```
AuthorizedKeysFile /var/ssh/%u
```

Of course you will need to create directories for the users under /var/ssh which only they can write to. Users will also need to create separate authorized_keys files for every server. This differs from many users' behavior of setting up a single authorized_keys file for all servers (since it is mounted by NFS). While this has some overhead, it once again encourages the principle of least privilege in that only machines for which the user explicitly requests passwordless connections will accept them.

Root Keys

Unrestricted SSH keys accepted by root are extremely powerful and should be avoided. Administrative users should generally use their own credentials to log into a server and then use sudo to gain root access there. Automation scripts that require remote root should use command-keys, which will be discussed further in "Command Keys." To enforce this, the PermitRootLogin option in sshd_config should be set to forced-commands-only.

Known Hosts

SSH provides powerful features to prevent server spoofing and man-in-the-middle attacks. Most notable is the use of public keys to strongly identify servers. This technique is not fool-proof however. SSH keys cannot be signed as X.509 certificates are, so unless you've received the server key from a trusted source, you have no way to know that the key is legitimate. There are three primary ways to get server keys: LDAP, centrally managed ssh_known_hosts and user-managed known_hosts. We will also briefly discuss using X.509 server certificates with commercial SSH.

Commercial SSH allows server keys to be centrally stored in LDAP, which is generally easiest to

[3]This paper uses the OpenSSH filenames and formats for configuration files. Commercial SSH uses slightly different file names and in some cases formats.

[4]Computer Incident Advisory Capability, CIAC Notes 95-07, "NFS export to unprivileged programs." See http://ciac.llnl.gov/ciac/notes/Notes07.shtml .

manage. OpenSSH and most free Microsoft Windows clients (such as Putty) cannot retrieve server keys from a central LDAP server, but for installations using commercial SSH, managing the server keys centrally is highly recommended. Whenever a server key is generated, it should be added to LDAP. For environments with multiple networks supported by different organizations, or for dealing with servers outside of your environment, commercial SSH supports multiple LDAP servers.

All UNIX SSH clients support a file called ssh_known_hosts, generally stored in /etc or /etc/ssh, which contains the official list of server keys. This file must somehow be distributed to all clients.[5] This file could also be NFS mounted, but this reintroduces the NFS security problems discussed above. Even so, NFS mounting this file may be better than not managing ssh_known_hosts at all. Centrally managing ssh_known_hosts is generally only effective within an organization. Since there can be only one file and it needs to be read from disk, there is no good way to include other organizations' host keys. Furthermore, since the users must trust the provider of the central ssh_known_hosts file to provide legitimate keys, this file can only be accepted as far as trust extends within the environment (generally as far as the central support organization).

If a server is not listed in the central ssh_known_hosts, SSH will by default prompt the user to add the key to the user's known_hosts, stored in ~/.ssh. This is the least secure option, since the user has no good way to determine the authenticity of the key. Once a key has been added to the user's known_hosts, however, SSH will warn the user if a server ever responds with a different key. This could indicate that a machine is being spoofed. Unfortunately it could also mean that the machine has been legitimately replaced. In environments where this is common, users have no good way to determine whether the warning is legitimate. To avoid these problems, it is highly recommended that ssh_known_hosts be centrally managed rather than rely on users' known_hosts.

Failing to centrally manage ssh_known_hosts creates special problems for automation scripts. Since scripts have no way to respond to the new key, they will fail if the key changes. This is a good thing in that it protects scripts from machine-spoofing, but it does create administrative headaches when scripts start failing due to a key change. Once again, the best solution to this problem is central management of ssh_known_hosts.

Commercial SSH improves this situation by allowing servers to use signed X.509 certificates rather than SSH keys. Since these keys are signed by a Certificate Authority, clients can rely on their authenticity without having all the keys in advance, greatly simplifying the administrative overhead of key management. Since most free clients (including OpenSSH) do not support these certificates, they are most useful in a completely commercial SSH environment, but in such an environment they are highly recommended as an alternative to ssh_known_hosts or LDAP.

Hardening Sudo

Unrestricted sudo effectively creates additional root-equivalent passwords for an attacker to guess or steal. Each administrator's password must now be protected with the same care as the root password. There are two approaches to mitigating this risk. Sudo-enabled administrative accounts can be separated from the administrator's regular account. Doing so will greatly reduce the opportunities for an attacker to steal the sensitive password. Alternately, sudo can be compiled to work with several one-time password systems such as OPIE, S/Key and SecurID. Deploying such systems is non-trivial, can be expensive and is beyond the scope of this paper.

Sudo has a significant security flaw in its default configuration that permits hijacking in which an attacker can make use of the victim's sudo privileges without the victim's password. Sudo uses tickets, files that are created to only require a user to enter her password at certain intervals. By default these tickets are created on a per-user basis, so if the user is logged on multiple TTYs on the same host, her ticket is valid for all of them. While modestly convenient, this is a significant security hole. If an attacker is able to run an arbitrary process as the victim user, then the attacker can piggyback on the victim's sudo privileges even without the victim's password. When the victim uses sudo, the attacker then has a five minute (by default) window to use sudo without a password. Coupled with the NFS authorized_keys attack discussed above, this is a very significant attack against administrative users.[6]

There is a complete but inconvenient solution to this, and an incomplete but fairly easy solution. The complete solution is to turn off password caching entirely, either by compiling with --with-timeout=0 or by setting passwd_timeout to 0 in the central sudo configuration file, /etc/sudoers. Doing so completely closes this particular attack, but strongly encourages system administrators to use a root shell to avoid retyping their passwords repeatedly. Since root shells cannot be easily logged, this is a significant auditing trade-off.

The less drastic solution is to compile sudo with --with-tty-tickets or set tty_tickets to "on" in sudoers. This will create a separate ticket to each user/TTY combination, stopping an attacker from piggybacking on the ticket in many cases. This is not a complete solution, however. The attacker can still attack the victim's login scripts to have the attack happen within the

[5] This works for managed UNIX clients, but has no good parallel for Windows clients.

[6] ssh-agent [OSSH] can be similarly attacked in order to make use of another user's SSH key. This seldom impacts automation tools because they are less likely to use ssh-agent, but it is worth keeping in mind for administrators.

victim's TTY. The attacker can also attempt to login to the server immediately after the victim logs off. On many operating systems (including Solaris, *BSD, and Linux) the attacker will often be allocated the same TTY as the victim had, and the ticket may still be valid. This latter attack can be mitigated with logout scripts that run "sudo -k" to destroy tickets, but it can be challenging to ensure that all administrators run this logout script. So turning on TTY tickets is better, but to completely close this hole, password caching has to be turned off.

It is very difficult to manage sudo such that users cannot escalate their privileges. This will be discussed further in "Controlling Sudo."

Limiting Privilege with SSH Command Keys

The most significant way to limit the power of an SSH key is to apply a command restriction. When a user connects using an SSH key with a command restriction, or a command key, a pre-defined command runs rather than providing the user with a shell. Applying this to root access, along with setting PermitRootLogin to forced-commands-only,[7] provides a powerful way to control automation tools. If the automation tool runs as a non-privileged user and only has access to a particular root command key, then that tool can get the root access it needs while reducing the ability to subvert it into performing arbitrary actions as root.

For most of the examples in this paper, we will consider the same simple task. We will change Apache's ErrorLog entry on a remote host to include the current month and restart Apache. This is a somewhat contrived example, since this would generally be done in simpler ways, but it demonstrates some of the main issues. The script we wish to run, update_errorlog, is shown in Figure 1.

To create a command key that runs update_errorlog, first create a keypair on the source machine:

```
$ ssh-keygen -t dsa -f errorlog_key
```

You now have a public key called errorlog_key.pub and a private key called errorlog_key. Prepend this with your command restriction and append it to ~root/.ssh/authorized_keys on the target machine. The format is as follows:

```
command="/usr/bin/update_errorlog"
                          [public_key]
```

[7]This only allows root to accept command keys, so there cannot be root-level SSH login keys.

Now login using the new key and the script will run:

```
$ ssh -i errorlog_key \
              root@target.example.com
```

Non-root Keys

Many remote functions do not require root access at all. By creating special users for these functions and providing them distinct SSH command keys, attackers who are able to steal the key will have extremely limited access.

This can be combined with sudo to provide functionality very similar to root command keys. By granting the special user specific sudo privileges, it is possible to create scripts that use root precisely when they need it and no more. As an example, we'll run update_errorlog (Figure 1) using a non-root SSH key.

On the target machine, create a new group apacheconf that can write to httpd.conf. We don't want to use the apache group itself, because httpd should not be allowed to write to its own configuration files (otherwise a security flaw in Apache could be used to reconfigure Apache). Use a low-numbered GID to help distinguish it from user accounts. Put httpd.conf into the apacheconf group so that our new group can manage it without root access.

Now create a new user, updatelog, to run update_errorlog. Put it into the apacheconf group and give it a low-numbered UID to help distinguish it from user accounts.

Our update_errorlog (Figure 1) script now needs a small modification, adding "sudo":

```
[...]
sudo /usr/sbin/apachectl graceful
```

Edit sudoers to grant the errorlog account permission to run "/usr/sbin/apachectl graceful".

Finally, set up a command key as we did in the "Command Keys" section, but instead of making it a root SSH key, make it an SSH key for errorlog.

We can now restart update httpd.conf from source. example.com:

```
source$ ssh -i errorlog_key \
        errorlog@target.example.com
```

Originator Restrictions

Keys (both regular keys and command keys) can be further restricted to specific originating hosts using the "from" option in authorized_keys. For example:

```
from="*.example.com,*.example.net"
     ssh-rsa AAA.oeTp0=rnapier@adminhost
```

```
#!/bin/sh
PATH=/bin:/usr/bin:/usr/sbin
date=`date +%F`
# Rewrite httpd.conf
perl -eip "s!^ErrorLog(.*)!ErrorLog /var/log/error_log.$date!" \
   /etc/httpd/conf/httpd.conf
# Restart Apache
apachectl graceful
```

Figure 1: update_errorlog.

This key will only permit connections from machines within the example.com or example.net domains. The "from" option accepts a comma separated list of canonical names or IP address including wildcards. Note that rnapier@adminhost is only a comment to give a hint where this key was created and has no impact.

Originator restrictions based on DNS names are reliant on trustworthy name services, but this still greatly increases the complexity of attack. The attacker must already have stolen and possibly cracked the private key, and then will still have to poison or compromise DNS in order to make use of that key.[8]

Other Restrictions

Most extended SSH features can be turned off on a per-key basis. This includes X-forwarding, port-forwarding, PTY-generation[9] and similar features. It is generally a good idea to turn off any features you don't need. For example:

```
no-port-forwarding, no-X11-forwarding,
    no-agent-forwarding,no-pty ssh-rsa
    AAA...oeTp0=rnapier@adminhost
```

Controlling Sudo

The Pitfalls of Limited Sudo

Using sudo to give limited access to root is a very tricky proposition, since the most obvious sudo configurations can be easily escalated to unlimited root access. As we discussed in the Introduction, even if you trust the user not to do this, you also have to trust the attacker who gains access to the user's password (or subverts sudo in some other way).

Some exploitable situations include:

- Permission to run commands in a user-writable directory.
- Access to chmod (even more easily exploitable with access to chown or cp)
- Access to any command with shell-outs (vi, emacs, ed, edit, more, less, find), though version 1.6.8 promises to help here
- Access to any command that can write (especially append) to an arbitrary file (vi, emacs, ed, edit, tee, less)
- Access to root's crontab or atjobs (crontab, batch, at)
- Any command that honors PAGER, EDITOR, or VISUAL (man, less, more)
- In some cases, any command that can read an arbitrary file (cat, less, more, tail). These can be used to get /etc/shadow for offline cracking, or can be used to read other protected files
- Access to sudo itself as root. This allows attacks like "sudo sudo /bin/sh". There are options to prevent this (--disable-root-sudo at compile time, or unsetting root_sudo in sudoers), but these are fairly weak protections meant to stop administrators from circumventing a !SHELLS entry. If you need these options, then you're probably allowing so many other commands (like those above) that an attacker can easily gain a root shell anyway.

With the release of sudo 1.6.8, two new features have been added that make limited sudo somewhat easier to implement. A common sudo need is to allow the editing of a protected file. This has historically been very difficult to provide in a controlled way without writing wrapper scripts. A user who is allowed to run an editor as root can almost certainly modify arbitrary files and trivially gain shell access. The new "-e" option to sudo, also accessible by running sudoedit, fixes this. It makes a temporary copy of the target file that is owned by the user. The user is then provided the editor of their choice, but since they are still running under their own userid there is no security issue. When the editor exits, sudo will replace the original file with the temporary copy. In the past, some administrators have written scripts to do just this, but moving this functionality into sudo itself should make things much easier. To allow a user to use sudoedit, treat it like any other command, but don't give a full path to it. The alias "sudoedit" represents either sudoedit, or "sudo -e". By appending a filename, you can restrict the user to editing particular files. For example:

```
rnapier host=(root) sudoedit /etc/httpd.conf
```

Another major improvement in 1.6.8 is the addition of a NOEXEC option.[10] On operating systems that support it,[11] the NOEXEC option will prevent a command run under sudo from calling exec() itself. This will prevent the shell-outs that provide trivial root shells from so many commands from editors to pagers. Given the newness of this technique, only time will tell how effective it is in practice.

The solution to providing limited sudo is single-purpose wrappers, small scripts written to do exactly what is required. By providing sudo access to just these wrappers, least privilege can be much better achieved.

For example, let's consider a script mysqllog, which prompts the user for her password, validates it against /etc/shadow, and if successful, displays /var/log/mysqld.log. This log file is owned by the mysql user and group-owned by the mysql group. It is only readable by user and group.

[8]SSH protects clients from connecting to the wrong server through host keys, but it doesn't protect servers from hostile clients. If a user shows up with the correct user key, no client host key checking is done. Even with the "from" restriction, only the DNS name is checked, not a host key, since there often will be no host key for a client.

[9]Many UNIX commands, most notably ls, have different newline handling if there isn't a PTY. If your tool can't handle this, you may need to allow PTY creation.

[10][SUDO], sudoers man page, "NOEXEC and EXEC."

[11]This includes at least SunOS, Solaris, *BSD, Linux, IRIX, Tru64 UNIX, MacOS X, and HP-UX 11.x. It does not work on AIX and UnixWare. [SUDO]

```
#!/usr/bin/perl -w
use strict;

sub error { print @_; exit 2 };

if( $> != 0 ) { error "Must run as root\n"; }

my $good_pwd = (getpwuid($<))[1] or error($!);
chomp( my $test_pwd = <> );
if (crypt($test_pwd, $good_pwd) ne $good_pwd) {
   exit 1;
} else {
   exit 0;
}
```

Figure 2: checkpass.

```
#!/usr/bin/perl -w
use strict;
my $user = getpwuid($<);
print "Password: ";
system( '/bin/stty', '-echo' ); # Don't echo
my $password = <>;
system( '/bin/stty', 'echo' ); # Do echo
print "\n";

open( CHECKPASS, '|-', '/usr/bin/sudo',
        '/home/rnapier/checkpass' )
   or die $!;
print CHECKPASS $password;
close CHECKPASS;
if ($? == 0) {
   system( '/usr/bin/sudo /bin/cat',
             ' /var/log/mysqld.log' );
}
else {
   print "Bad password.\n"
}
```

Figure 3: mysqllog.

Since mysqld.log is group-owned by mysql, the script will need to have access to that group. It also seems to need root privileges in order to access /etc/shadow. The obvious solution is to simply make it a setuid root script, but this would give it far more access than it needs. In fact, this script doesn't actually need to be able to read /etc/shadow; it only needs to be able to verify that a given username/password combination is valid. Carefully stating your privilege requirements is the first step towards achieving least-privilege.

As before, we'll create a user, mysqllog, to run this script and edit sudoers to give it permission to run "checkpass" and "/bin/cat /var/log/mysqld.log".

The checkpass script is listed in Figure 2. It reads a password for the current user from STDIN. It then exits with a 0 to indicate a good password, a 1 to indicate a bad password, or a 2 to indicate an error. We pass the password in on STDIN because command line parameters can be seen in the process table by all users. Note that this script can only validate the current user, not an arbitrary user. Once again we keep to least privilege.

The code to perform our task is shown in Figure 3. We make it setuid to the mysqllog user we created earlier. Now arbitrary users can run this script, enter their password, and get the contents of mysqld.log. Even if an attacker can find a bug in the script, the privileges that can be exploited are very limited.

Setuid/setgid vs. Sudo

Setuid scripts can check the calling user to determine whether they have rights to run this script (generally by checking group membership). This is convenient for the authorized user, because she is saved the trouble of typing "sudo." An example of such a check in Perl is given in Figure 4.

```
#!/usr/bin/perl -w

my $group_wheel = getgrnam( 'wheel' )
                      or die;
if( $( !~ /\b$group_wheel\b/ ) {
   die( "Must be part of wheel".
    " group to run this script.\n" );
}
```

Figure 4: Checking group membership in Perl.

Alternately, privilege-requiring scripts can be executed using sudo. This has the advantage of providing centralized accounting of all privileged users and reducing the complexity of the scripts.

Non-root Sudo

One should always consider when using sudo whether the user needs root access or whether access to a non-privileged user like apache or jabber might be sufficient. Sometimes changing the ownership or group of configuration or log files is enough to allow less-privileged accounts to manage them. Be careful with this, however. Many services like Apache should not be run under a UID that can write to their configuration files. Doing so could allow a minor compromise to be escalated into a larger compromise by allowing the server to be reconfigured by an attacker. That said, there is no reason that Apache's configuration files can't be owned by an apacheconf user and administrators given appropriate sudoedit privileges to that user.

Setuid/setgid with Sudo

Setuid/setgid can be combined very effectively with sudo. For example, the script can be setgid to a special group. This group can then be given root sudo privileges to run specific single-purpose wrappers. The script can then use sudo to execute these wrappers to escalate to root privileges precisely when needed, and only for precisely what is needed. Furthermore, since the single-purpose wrappers are not themselves setuid, they can only be called indirectly, by already-privileged processes. This helps prevent an attacker from passing them unusual parameters, making them less susceptible to security coding flaws.

As an example, we can achieve the same functionality as in update_errorlog (Figure 1) on our local machine using a setuid script (Figure 5). As in the "Non-root keys" section, we'll set up an errorlog user, including its sudo privileges, and make the script setuid to errorlog. When you run update_errorlog it will then update httpd.conf and restart Apache as long as you are in the wheel group.

Similar techniques can be used with SSH command-keys or CGI scripts that are run under specific user IDs.

Setuid/Setgid Best Practices

Non-root Setuid

"Setuid" is sometimes confused with "run as root," but this need not be the case. Setuid can be used to run a command as any particular user by chowning the file to that user.

As with sudo, always consider whether your setuid script really needs root access or just access to a special user. For example, if a script needs access to a file containing a password, there's no reason that file needs to be owned by root. It could be owned by any non-user account.

Reducing the number of setuid-root scripts reduces the number of ways attackers can exploit coding errors to obtain root.

Setgid

In some cases, setgid can be even more useful than non-root setuid. As we saw in the "Non-root keys" section, creating a group to manage configuration files such as for Apache can help isolate access to these files from root access. Setgid scripts can grant users access to these protected files, while still preserving the user's own privileges (such as access to their home directory) without any special handling of effective UID.

Setuid Script Obfuscation

Setuid scripts in languages like Perl and Python present a special problem. They have to be readable by the user, giving an attacker an opportunity to study them looking for security flaws to exploit. The attacker may even be able to copy the script to another machine to test possible exploits offline.

Compiled programs do not generally have to be readable by the user; they only require that the executable bit be set. So when writing setuid and setgid scripts in interpreted languages such as perl or python, there is some value to creating a small wrapper in C, as shown in Figure 6.

```
#include <stdio.h>
#define CMD "/usr/local/protected/myscript"
main(ac, av)
  char **av;
{
  char error[80];
  execv(CMD, av);
  snprintf( error, sizeof( error ),
                    "Unable to run %s",CMD );
  perror( error );
  exit( 1 );
}
```

Figure 6: myscript.c setuid wrapper.

In the above example, /usr/local/protected should only be readable by the setuid user (often root), and myscript should be replaced with script filename.

Keep in mind that this is an obfuscation technique, not a security technique. If your script had no security flaws in it, then this technique wouldn't be needed and using this technique doesn't prevent an attacker from exploiting your script's security flaws. It just makes finding the flaws harder.

While languages like Perl and Python have special handling to make setuid scripts "safe" (though readable by the user), it is not trivial (or even possible on some older platforms) to make Bourne and similar shell scripts setuid safely. Most operating systems don't even allow this anymore.[12] These will absolutely require a wrapper script, though it would be wise to

[12]Shell scripts are subject to environment attacks including manipulation of PATH or IFS, and timing-based attacks based on moving links around between the time that the script is started and the script is read. Some of these have been addressed in modern versions of UNIX-based operating systems, but because of Bourne shell's reliance on exter-

```
#!/usr/bin/perl -w
use POSIX qw(strftime);
my $group_wheel = getgrnam( 'wheel' ) or die;
if( $( !~ /\b$group_wheel\b/ ) {
  die( "Must be part of wheel group to run this script." );
}
my $file = '/etc/httpd/conf/httpd.conf';
my $date=strftime("%F", localtime);
if( -e "${file}.bak") { unlink( "${file}.bak" ) or die "$!" }
rename( $file, "$file.bak" ) or die "$!";
open( INFILE, "$file.bak" )  or die "$!";
open( OUTFILE, ">$file" )    or die "$!";
while( <INFILE> ) {
  s!^ErrorLog(.*)!ErrorLog /var/log/error_log.$date!;
  print OUTFILE;
}
close INFILE  or die $!;
close OUTFILE or die $!;
system( qw(/usr/bin/sudo /usr/sbin/apachectl restart) ) == 0 or die;
```

Figure 5: update_errorlog in Perl.

write setuid scripts in a language like Perl or Python in any case. Bourne shell and its cousins rely very heavily on the operating environment and external programs and so are much harder to adequately secure in a setuid context.

Perl in particular provides taint checking, which helps programmers keep track of what data could have been influenced by the user. Setuid Perl scripts automatically turn on taint checking. If you use a C-wrapper as above, Perl will no longer automatically turn on taint checking, so you should do so by passing "-T" in the perl invocation.

Finally, whenever possible, make setuid and setgid programs unreadable by anyone but the owner.

Best Practices in Handling Privileged UID/GIDs

Ideally all "special" UIDs and GIDs should have a consistent numbering convention. Generally, numbers under 100 (or 1000 for larger systems) should be reserved for these special IDs.

Special UIDs should not generally permit direct login. They should not have a valid password or shell.

It is often convenient for administrative staff to belong to special GIDs so that they can manage configuration or data files directly without needing further access (such as sudo). This is particularly useful for allowing non-root users to administer particular parts of the system.

Odds and Ends

Sticky Bits

Setting the sticky bit on a directory allows users to write files that other users cannot remove, even though the directory is world writable. In some cases this can get rid of the need for setuid user scripts to write protected files. /tmp is a good example of where this is used. Setting the sticky bit is done as follows:

```
chmod o+t directory
```

Setgid Directories

Setting a directory setgid will cause files created there to belong to the same group as the directory rather than the user's primary group. In some cases this can get rid of the need for privileged daemons that need to read things created by users.

Combining this with the sticky bit is an effective way to create a drop-box location for a non-privileged daemon. Users can put things into this directory, but they can't list the entries in the directory (since we won't add the directory read privilege), and they can only remove their own files (because we'll set the sticky bit).

Create a directory "drop" and set the sticky and setgid bits:

```
# chown myservice:myservice drop
# chmod u=rwx,g=rwxs,o=wxt drop
```

This means that anyone can drop files into the drop directory, users can't modify each other's files and the myservice user doesn't need any special privileges to manage files dropped into this directory.

Web Applications

By default, web applications run as the web user. On a system with multiple web applications, each application will have access to the others' data in this configuration. By creating separate accounts for each application, they can then be separated using suexec, an Apache feature that causes CGI programs to run under user accounts rather than as the web server. This can protect application data from exploits against other CGI programs as well as the web server itself.

Recommendations for the Future

There are several features that would help large installations manage sudo and SSH better. Some of these are currently possible with custom work, but they should be integrated better into the products.

- SSH should be able to get its authorized_keys information easily from LDAP or Active Directory rather than the user's .ssh directory. This would allow the list of authorized keys to be protected from NFS attacks without resorting to local configuration files on each host (as described in "Hardening and Managing SSH"). To maintain least-privilege, LDAP keys should be assignable to specific servers (rather than being globally accepted), and would need to allow restrictions such as "from" and "command". The existing SSH LDAP solutions[13] allow X.509 certificate authentication of users, but do not fully replace the authorized_keys file.
- SSH needs better integration with one time passwords (OTP). In particular, it should be possible to configure keys that allow an interactive shell to require one-time passwords while not requiring this for command keys. Interactive shells are extremely powerful and should always have a human available to enter the OTP. Command-keys are very restricted and generally won't have a human available to enter the OTP. This is difficult to implement with the current SSH tools, which generally requires all-or-nothing use of OTP.
- Sudo needs to be able to get sudoers configuration from LDAP. This would make it much easier to integrate with a Role Based Access Control (RBAC) system or other centralized account and authorization systems. It is currently possible to generate a sudoers file out of LDAP with custom tools, but this is cumbersome and creates

nal programs for most handling, it is very difficult to protect yourself from all of them. Most modern UNIX-based operating systems do not permit setuid shell scripts. See the UNIX FAQ Question 4.7 at http://www.faqs.org/faqs/unix-faq/faq/part4 for more information.

[13]LDAP is managed by the "Certificate Authentication" feature of commercial SSH and the "OpenSSH LDAP Public Key Patch" (http://ldappubkey.gcu-squad.org/) for OpenSSH.

a delay between when LDAP is updated and when the change takes effect.
- A tool that would automatically determine trust relationships created by sudo and SSH and display this in a consolidated format (such as a directed graph) would be extremely valuable.
- Sudo should use TTY tickets by default and optionally clean up old tickets automatically whenever sudo is run.

Availability

OpenSSH is freely available under the BSD license from the OpenBSD project. Their website is available at http://www.openssh.org. At the time of this writing, the currently available version is 3.8.

SSH Secure Shell, discussed in this paper, has been replaced by SSH Tectia. Both are commercial products available from SSH Communications Security (http://www.ssh.com). Where this paper refers to the commercial product, it is written to SSH Secure Shell version 3.2.9. At the time of this writing, the most recent version is SSH Tectia 4.1.

Sudo is freely available and maintained by Todd Miller (Todd.Miller@courtesan.com) at http://www.courtesan.com/sudo. At the time of this writing, the most recent version is Sudo 1.6.7p5, though some features of the upcoming 1.6.8 are discussed in this paper.

Conclusion

In this paper we have established the importance of the principle of least privilege to the overall security of an environment, by reducing the avenues of attack and the extent that any particular attack can compromise the system as a whole. We have discussed problems with the techniques that may currently be used in many environments including unrestricted SSH keys for automation tools and setuid tools with excessive privileges. Finally we have provided techniques and examples of how to apply least privilege to real-world automation problems, including restrictions on sudo and SSH, wrapper scripts, setgid and sticky directories. While these techniques are useful and important, even more important is the philosophy behind least privilege. By constantly asking ourselves what the minimum set of privileges a particular operation needs, and challenging ourselves to reduce and compartmentalize those privileges, the security of our environments will not only improve, but become pervasive.

Author Information

Robert A. Napier is a founding member of the Corporate Information Asset Protection team for Cisco Systems where he trains internal groups on classifying and protecting sensitive information with special focus on technical safeguards. He is a member of the GCUX board and also holds IAM and CISSP certifications. He is a co-author of *Special Edition: Using Linux 6e* and a contributing author to *Special Edition: Using KDE* and *Red Hat Linux Installation & Configuration Handbook*. He can be reached electronically at rnapier@employees.org.

References

[SUDO] Miller, Todd, *Sudo Main Page*, http://www.courtesan.com/sudo, 2003.

[OSSH] OpenBSD, *OpenSSH Manual*, http://www.openssh.org/manual.html, 2004.

[SSH] SSH Communications Security, *SSH Secure Shell for Servers Version 3.2.9 Administrator's Guide*, http://ssh.com/support/documentation/online/ssh/adminguide/32, 2003.

Experience in Implementing an HTTP Service Closure

Steven Schwartzberg – BBN Technologies
Alva Couch – Tufts University

ABSTRACT

One ideal of configuration management is to specify only desired behavior in a high-level language, while an automatic configuration management system assures that behavior on an ongoing basis. We call a self-managing subsystem of this kind a *closure*. To better understand the nature of closures, we implemented an HTTP service closure on top of an Apache web server. While the procedure for building the server is imperative in nature, and the configuration language for the original server is declarative, the language for the closure must be transactional; i.e., based upon predictable and verifiable atomic changes in behavioral state. We study the desirable properties of such transactional configuration management languages, and conclude that these languages may well be the key to solving the change management problem for network configuration management.

Introduction

HTTP servers are complex applications. In crafting a valid configuration file for a server such as Apache, there are many options, often with cryptic names and unclear meanings. Choices for many options seem not to matter to the end-user, e.g., the exact locations of content hierarchies within the filesystem. Choices for other options have critical effects, such as whether to allow CGI programs within a particular directory to execute. Simple typos in the configuration file can be difficult to locate and have unpredictable results. Thus, to assure reliable service, many configuration changes must be made by an experienced system administrator.

We manage an HTTP server cluster where the majority of responsible system administrators and content providers have historically been relatively inexperienced students. As a result, there has been considerable service downtime due to misconfiguration of the server, giving content files inappropriate names or MIME types, inappropriately protecting content, and even allowing servers in the cluster to differ in configuration.

Content providers often make serious errors in naming files and setting permissions for HTTP content. Either content is protected too restrictively to be available, or content protections are permissive enough to pose a security risk. Users also have difficulties ensuring that files have extensions that match their content. Typical examples include inadvertently exposing private contents of scripts by giving them incorrect extensions or filing them in an inappropriate directory, or making content directories world-writable, thus posing a security risk.

Naive editing of HTTP configuration files can also cause unexpected and costly downtime for web servers. When many virtual domains inhabit one server, one configuration error can be catastrophic, as it will bring down all of the domains. In practice, it can take a lot of time to recover from an error if changes have not been carefully tracked. At times, low-traffic virtual domains have been down for weeks due to undocumented changes that had hidden effects.

An HTTP Closure

We seek to solve this problem by surrounding our HTTP service with a closure, as described in [17]. A closure is a self-managing component of an otherwise open system. We intended our closure to:

1. Allow relatively untrained users to reliably create relatively complex configurations involving virtual domains and aliases.
2. Allow reliable creation and deletion of virtual domains.
3. Ensure appropriate protections and MIME types for content.
4. Protect against unauthorized changes to content or configuration.

To accomplish this, we had to make a radical departure from the way HTTP servers are usually administered.

At the beginning, we were inspired by several related projects. DryDock [20] is a content-management system that allows the submission of web pages to a web server, after they have checked by a human being. While it provides some desired features, including content validity checking, DryDock is more of a content checking and approval method than a web server configuration tool. TemplateTree II [30] can help configure the webserver configuration file, by automatically filling in blanks in a pre-determined template, but seems to stop short of being able to handle advanced features such as virtual domains. Each of

the virtual domains requires the addition of a new template to the file. Charlie [38] is a content-mapping webserver in which URL's are explicitly mapped to files by a declaration file, and service is not determined by filesystem structure. Our initial goal was to combine these ideas into a reasonable HTTP closure in which:

1. Content is specified by mappings between URLs and files (Charlie).
2. The appropriate configuration is generated from intermediate data (TemplateTree II).
3. Content is checked for type validity before being published (DryDock).

Our hope was that the resulting synthesis would be easier to use than any of its predecessors.

Recently, others have attempted part of this process independently. The Virtualmin [9] environment within the Webmin [8] web-based administrative environment solves the problem of defining virtual servers neatly, but does not deal with the problems of content management and assuring that content is provided with correct MIME types, etc. Both Virtualmin and Linuxconf [19] support dynamically changing the modules that are loaded into Apache, though to our knowledge they do not address the dependency issues we discovered in trying to accomplish this. A third management environment, Comanche [32], was unavailable to us except in source form at time of writing, and we lack knowledge of its capabilities.

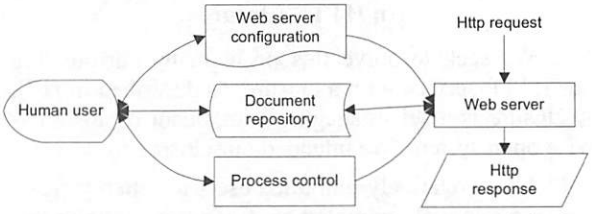


Figure 1: Typical interactions between a user/administrator and web server, where arrows indicate information flow.

HTTP servers are typically administered through direct access to configuration files and documents (Figure 1). One edits a server configuration file directly and then places content directly into directories that the server should expose to the outside world. The configuration file serves not only as a means of control, but also as documentation of defaults and other performance characteristics of the web server. Without fairly complete knowledge of the contents of this configuration file, it can be difficult to publish content. Since the document repository is edited directly, users must also have a good grasp of file protections within the server environment. Current approaches to this problem include simplified graphical user interfaces that expose only part of the capabilities of the configuration file [9, 32].

The configuration of the Apache web server is described by the contents of several files (Figure 2), where arrows indicate couplings between data in differing files or structures. In order for the server to respond correctly to a request, several parts of the configuration must agree in intent. For example, in the figure, to answer the request for URI http://www.foo.edu/, it must be true that:

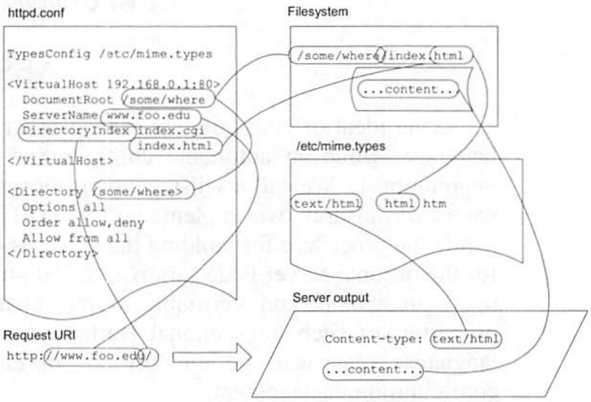


Figure 2: Configuration constraints required in order to answer a request properly on an Apache web server, where lines indicate required agreements between data values.

1. www.foo.edu is a valid virtual domain.
2. www.foo.edu maps to the server's address.
3. www.foo.edu's content is stored in the directory /some/where.
4. It is permissible to publish the content of the directory /some/where.
5. The request is for a directory rather than a file.
6. In returning data for a directory, one first checks for an "index file" that represents directory content.
7. index.html is an appropriate file.
8. index.html exists in the directory /some/where.
9. index.html is readable to the web server.
10. index.html has MIME type text/html because of the .html extension.

If any of these assertions is not true, the request fails. In the figure, many of these constraints are indicated by lines between configuration data that must agree in value.

The net effect of this scheme of constraints is that an Apache server can be quite difficult for a novice to administrate. There are several reasons for this:

1. The effect of a particular declaration depends on other declarations; one needs to understand the global configuration in order to understand the effect of a local declaration.
2. The configuration language – in an attempt to be easy to type – is filled with seemingly convenient defaults that make a configuration file difficult to interpret.
3. Often several distributed declarations determine whether content is provided correctly. For example, in order to serve a directory, one must

specify its protections as a directory, its mapping as a URL, and its MIME type mapping (if different from the default).

To simplify this process, we took direct control of content directories and configuration file away from the user. These are instead controlled by an intervening layer that mediates between user and server (Figure 3).[1] The user interacts directly only with an *image* of the document repository and a command interpreter. This interpreter keeps track of its state and maintains a private document repository of its own to which a user does not have access. User commands cause copying from the user's space into the closure's space in order to publish a document.

It might seem that we have just made the process of publishing more difficult, but in fact we have made it much less error-prone. At several steps during the publishing process, validity checks are made to the document and configuration requests. These checks include:

1. Does the name of each virtual domain correctly map to a valid interface via DNS?
2. Does each file's MIME type roughly agree with its content, as indicated by file magic numbering?
3. Do HTML files contain correct HTML?

Once a document passes these validity checks, it is published reliably, because the closure will take care of placing it in a proper location and protecting it so that the web server can see it. Also, a validate command checks that the web server configuration and all content have not been edited by unauthorized people, by comparing cached MD5 checksums against checksums of current data.

This closure consists of three components:

1. A *setup script* that initializes the closure on a prebuilt server.
2. An *agent* that interprets the command language and makes changes in configuration over time.
3. A *command language* for describing changes to make in service.

[1]Some authors would call this "middleware." "Middleware" is perhaps one of the most abused terms in the modern computing lexicon. As we provide not an API but instead a message-passing interface, the term "middleware" does not seem to accurately apply.

We chose to place our closure around an Apache web server running inside RedHat Linux 9.0. We assumed that the underlying system would be newly built and functioning on the network prior to invocation of the closure. In an actual closure, all systems with which the closure will interact must also be closures, in order to maintain overall integrity. Since this was our very first closure (which, in hindsight, was probably too ambitious), we had to settle for interacting with an already functioning system.

The initial "build script" determines a few aspects of pre-existing system configuration, such as where the Apache web server is located, whether certain applications are installed on the system, and other vital data necessary in order to construct the closure. The script then creates a managed structure on the disk that has restricted permissions. This structure contains a startup/shutdown script for the web server, a document root, a (private) space for storing closure logs and data files, and a default configuration. Data files include definitions of virtual domains, access rights for users and directories, MD5 checksums of configuration and submitted files, and boilerplate configuration segments describing defaults. Internal data is stored in the Perl Data::Storable format.

After the build script does its work, a command interpreter takes commands and modifies the resulting service automatically. This interpreter is a perl script that acts as a command line interface. This interface is responsible for all ongoing management of the HTTP service. In order to make the closure easy to use for both system administrators and end users (who may not be familiar with computer-related concepts), we decided to create a very limited command set that should be able to provide all the functionality necessary for a typical multiple-domain server.

Most of this process is straightforward; difficulties arose mainly in designing an appropriate command language with which to converse with the closure. Coming up with an appropriate language took considerable thought and required nontrivial changes in the way we think about server configuration as a process.

Command Language 1.X

Our initial command language was patterned after the structure of Apache's configuration file

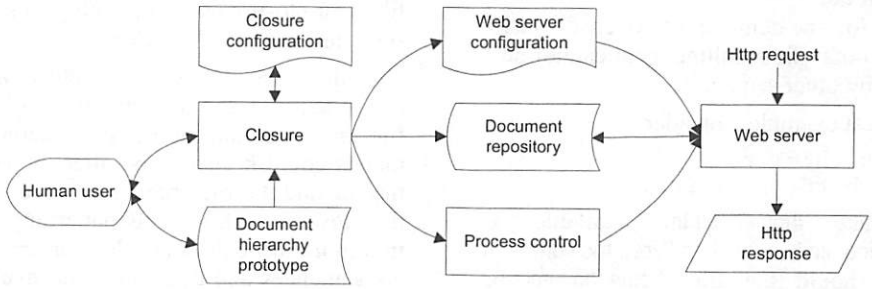


Figure 3: A closure mediates between the user and the Apache configuration, reducing complexity of the configuration process.

httpd.conf. We decided upon a minimal language with typical commands like the following:

```
assert foo.bar.com
```

declares a virtual host and readies it to serve content.

```
retract foo.bar.com
```

removes a virtual host from the server, along with all content that it provides.

```
post /home/prod
        http://foo.bar.com/products
```

makes the directory /home/prod on the current machine appear to be the web directory http://foo.bar.com/products.

```
post /home/couch/foo.html
        http://foo.bar.com/foo.html
```

makes the file /home/couch/foo.html appear as the URL http://foo.bar.com/foo.html. Each URL is associated with a unique directory in a private managed space, and each posted file or directory content is copied there to isolate it from further changes by the user. The type of the first argument – file or directory – determines what post does. In the case of a directory, post copies all subdirectories recursively.

```
retract http://foo.bar.com/products
```

removes any association between that URL and a content directory. It erases content previously associated with the URL via a post command.

```
retract http://foo.bar.com/foo.html
```

removes the mapping that results in content for the above URL. As described in [17], a web server is a mapping between URLs and content; the user need only specify that mapping and the closure takes over to assure it.

Ambiguity

At this point, progress on the project ground to a halt due to seemingly insurmountable difficulties within the command language. *Ambiguity* arose in the command language as a direct effect of *defaults* in the underlying httpd.conf, as well as default behavior for the corresponding HTTP protocol. This ambiguity has several effects:

1. The causal effect of many commands is unclear or determined by context.
2. One needs to understand the history of all commands in effect to know exactly what a single command will do.
3. It is possible for one command to override *part* of another, so that the resulting state cannot be reached via any other sequence.

As the simplest example, consider:

```
post /home/foo.html to
                http://www.foo.com
```

Should this make /home/foo.html available as http://www.foo.com/foo.html or as http://www.foo.com? In the latter case, should the file /home/foo.html be renamed to index.html or left alone in the directory? Then consider:

```
post /home/foo to http://www.foo.com
```

If /home/foo is a file, this has a potentially different effect than if /home/foo is a directory. But we cannot know which it is from the command itself. Finally, consider:

```
post /home/index.html to
        http://www.foo.com/index.html
post /home/index.cgi to
         http://www.foo.com/index.cgi
```

Which of these will be the directory index? The result of answering these questions in any reasonable way was that the effects of the seemingly simple command language were hideously complex to *document* and *understand.*

Creating content for a web server requires complete knowledge of its conventions, including which filenames have special meanings or interpretations. These defaults are set in httpd.conf. Without complete knowledge of the defaults, the user cannot create content properly. By abstracting the defaults into a command language rather than a file, we made the defaults invisible, and thus rendered the problem more difficult than before. We concluded that we had not simplified the problem of managing httpd.conf; we had actually made the management process more difficult than before!

Appropriate Closure Language

The key to these quandaries was to carefully describe desirable attributes of the command language and then redesign to these requirements. But we did not understand the optimal properties of such a command language, and only had the httpd.conf format as an example. Its properties include:

1. *Minimization of ink:* all that is unspecified has (reasonable) defaults.
2. *Hierarchy:* everything is laid out in a carefully designed multi-level hierarchy.
3. *Scoping:* the intent of commands depends upon the context in which they are entered.
4. *Ordering:* ordering of certain commands, including protections, changes intent within the configuration file.

These properties ease the loop of interacting directly with the configuration file, but do *not* ease the process of incrementally describing a configuration through individual and atomic commands. The design of httpd.conf presumes that the administrator has global knowledge of the contents of the whole configuration file. Our closure users, operating from outside the closure, have no such knowledge.

In effect, the syntax of httpd.conf had corrupted our thinking. Used to being able to look at the whole file to answer questions, we presumed that our commands could be patterned after the edits we make to the file and the file copying that we would do without the closure in place. This patterning made configuration more difficult rather than easier. In fact, the exact conveniences and defaults – that make httpd.conf easy to use when one is editing it directly – make a transactional language difficult to use.

To be easy to use, our command language has to have several somewhat different properties:

1. *Clarity*: the intent of a command should be immediately clear from its form.
2. *Independence*: to the extent possible, commands should be independent of one another to avoid conflicts, ambiguities, and difficulties in determining effects. In particular, global defaults should not be subject to change.
3. *Declarative syntax*: Each command represents a state to preserve, rather than an action to perform.
   a. Commands should be *idempotent*, i.e., repeating a command twice in a row should have the same behavioral effect as doing it once.
   b. Commands should be *stateless*, i.e., the behavioral effect of doing a command should not depend upon prior executions of the same command, even if these executions occurred in the remote past. A stateless command is always idempotent; statelessness is a stronger condition.

   Reducibility to assertions: Either an assertion is in effect or not; there is no such thing as being "1/2 in effect." A command that conflicts with a previous command undoes ("retracts") the effect of the conflicting command. Equivalently, any sequence of assertions and retractions is equivalent with a subsequence consisting of assertions alone.
4. *Representability*: at any time, one should be able to get an idea of all commands currently in effect, to understand global contents. Ideally, the representation should be a conflict-free description of the current state of the service, in terms of the unordered list of commands currently in effect. In particular, the representation of service should be free of retractions.

These properties actually arise from mathematical models that we will describe later. For now, it suffices to mention that statelessness and reducibility to assertions imply representability. The remainder of the requirements, clarity and independence, contribute to ease of use.

Statelessness means that a command does not depend upon prior invocations of itself to do its work. Stateless commands cannot be incremental in nature, but must deal with absolute quantities. For example, incrementing a counter is not a stateless command. The reason that statelessness is important is that the user may not have knowledge of prior commands or pre-existing configuration. While a stateful command may have indeterminate results, a stateless command has (roughly) the same effect regardless of when it is executed. The user need not remember anything in order to know what its effect is.

Representability means that at any time, one can describe the closure via the commands that are currently in effect, which is typically a smaller list than the whole sequence of commands since the closure was created. Reducibility to assertions means in addition that the commands currently in effect do *not* have to contain retract statements, because conflicts cause conflicting assertions to completely disappear from a representation.

Command Language 2.X and Beyond

These considerations caused subtle but profound changes in our command language that both resolve ambiguities and make it easier to use than making manual changes to configuration files and web content. We are currently in the process of implementing these changes.

1. Commands either *augment* or *retract* the effects of other commands. The effect of issuing a conflicting command is to retract the commands with which it conflicts. To assure this, we deprecated using post for files (except for indexes), and required its use on directories, where it recursively applies to subdirectories. In version 1, retracting a directory does not retract its subdirectories; in version 2, subdirectories are retracted as well.
2. When overriding the MIME type for a specific file, the command does not affect other files with the same extension. In version 1, a MIME-type override applied to all of the files of that type in a folder. This caused confusing changes in default MIME-types when new files were uploaded. To assure clarity of intent, the override is now specific to each single file.

Other profound changes are yet to be implemented. The above ideals for language imply that the indexing process for a directory should be *independent of its content.* One should be able to specify an index for an empty directory, or have a directory with no index. In the latter case, one gets a "permission denied" error instead of a directory listing. This effect is accomplished by adding the special keyword index to a post command for the index file.

Thus we plan to change the *concept* of index from being a file with the word index as its name, to being a file with *any* name that just happens to be bound to a directory as a listing operation. This index file can have any name, and be of any file type. This change provides a significant increase in flexibility over the old style, while disambiguating the most frustrating of problems when faced with the problem of insuring consistent operations. If no index file is selected, then either an error page is returned by default, or the security on the system can be made more lax and allow one to display the contents of the requested directory.

Critique

Our current closure does its intended job well, but there are many shortcomings. While several are simply new features to be implemented, some require a relatively deep rethinking of how we interact with systems that provide web content.

First, while the intent of a configuration can be represented by a list of commands, repeating the commands will *not* produce the exact same HTTP service. There is no guarantee that the source directories have not changed in the meantime. Repeating the same set of commands that created the service will instead result in an *updated* service based upon changes in the source directories. Indeed, what we seem to have implemented is the opposite of a content-staging system such as DryDock; we included no protection against inadvertent changes of the *source* repository.

Many issues for dynamic content have yet to be resolved. Various programs, such as Gallery [39], create and/or modify files within the web hierarchy by themselves, which constitutes performing operations outside of the closure. This kind of interaction is not supported in our model, and though it is tolerated, perhaps should not be allowed at all.

The closure deals very poorly with dynamic content stored within the document repository. Repeating commands used to set up a repository will erase any dynamic content created in the meantime within the repository by the action of CGIs. This content is not even *accessible* to the user unless exposed by the web server, and any files posted via the closure will be automatically made read-only.

It could be argued that this fascism is not a bug, but a *feature*; it strictly enforces utilizing external databases to store dynamic content rather than local server files. This is indeed the "best practice" for managing dynamic content, according to many web programmers.

In fact, we can find no reasonable solution to supporting this behavior of CGIs. If we allow dynamic content in document directories, it must be protected from subsequent post commands (that, in normal operation, will delete that content). But if we ignore such files, then the effect of post is not stateless.

Ideally, CGIs should use external data sources for dynamic content. The couplings between CGIs and external data sources should be managed by the closure itself, though the best language for accomplishing this is unknown. Another rather deep question is how to handle enforcing integrity constraints for data sources outside the closure, such as databases utilized by CGI scripts. If we do force all programmers to utilize databases, how do we assure that their scripts bind to functional and allowed data sources? While the Microsoft .NET framework makes this kind of checking easier by separating data source bindings from programs, no such solution exists for Linux and Apache.

Finally, this closure was our very first closure, and thus had no other closures with which to converse (unlike the ideal web service closure described in [17]). Thus our closure must check for external dependencies itself, and cannot correct deficiencies that it finds in its environment. For example, the assert command checks whether the domain being asserted points to this machine in name service, but cannot assure that by modifying the name server. Instead, it must refuse to perform the assert.

Future Work

Plenty of additional work needs to be done on this prototype before it is appropriate for production use. Among the most obvious extensions is to be able to handle ssl (HTTPS) traffic as well as HTTP. This will require improving how the closure deals with constraints. One tricky part of handling ssl, for example, is that one must enforce the constraint that only one virtual server bound to each address can have ssl capabilities. Currently, one must explicitly disable one ssl instance before asserting another.

Separating indexing from directory contents is a bit tricky, as Apache is designed to couple them together. Currently the closure simply uses whatever index file is present in each directory. Implementing the ideal indexing scheme – in which indexing is completely separate from directory contents – requires special care to avoid naming conflicts. This can be handled by storing the index in a name that will not be used otherwise (and cannot be easily entered as a URL), such as "-->index<--.__html__". To avoid confusion in choosing an extension matching the MIME type of the index file, this file is a simple CGI script that will read the real index content from a second cryptic filename "-->content<--.__html__", and display the contents, tagged with the appropriate MIME header.

There are also problems in dynamically managing the modules that Apache loads and requires. A true closure would need to analyze what the user desires from the web server and load the necessary modules by creating a dynamically generated list. We thought we could look into the directory where the modules are located and instruct Apache to load every one it finds as a starting step. We discovered quickly that certain modules are dependent upon others, that the order in which these modules are loaded is important, and that some modules conflict with and preclude the use of others. For example, mod_proxy_ftp.so requires mod_proxy.so to be loaded first or else loading will fail; likewise loading mod_dav_fs.so will fail if mod_dav.so is not loaded first. Using the current scheme for loading modules, there is no way to know in advance what dependencies, if any, a module actually has without trying to load it.

As a temporary solution, the list of modules to be loaded had to be made static, along with the order in which they are loaded. However, this means that the functionality of the closure is currently severely limited, as there are no commands to the closure that can expand the capabilities of the server.

Another ongoing issue is rights management. Currently, a user can only be granted rights to edit a

domain. In addition, we should be able to permit users to update sections of a domain without having rights to other sections. For example, we have started writing code that will allow couch to edit http://www.foo-bar.com/research, but restrict his access to other areas of the domain.

Other low level issues, such as insuring that there is adequate disk space for web content, need to be addressed, though this may need to be performed by talking to a disk closure that has yet to be written. Since varying devices have different access speeds, this could be a future concern, as some content may need to be delivered at a much higher quality of service (QoS) than other content, and one would want those pages to be stored on faster access devices.

Lastly, we would like the application to be able to handle not only a stand-alone web server; it should also be able to scale to a grid or cluster type configuration, which brings a whole new level of difficulties and questions, but ultimately a far more powerful and desirable closure.

Theoretical Background

While we were struggling with the implementation of the prototype closure, another struggle was going on at a different level. Clearly, our language evolved into something relatively useful from something relatively useless. But why do the above language principles work, and what mathematics underlies the design decisions we made? In this section, we explore some of the mathematical underpinnings of closure language design, and tie this work into other work on languages for configuration management. For the non-mathematically inclined, this section can be skipped without loss of continuity.

An overview of the results of this section is shown in Figure 4. Statelessness of individual commands leads to idempotence of sequences of commands. The ability to remove retractions of commands from a sequence and retain equivalent effect is called "reducibility to assertions." This property, in combination with a semantic model that determines preferred order of operations, makes a sequence of commands declarative in character. Statelessness of commands, reducibility to declarations, and a one-one correspondence between configurations and behaviors give rise to Cfengine-like convergence of the declarations, thought of as an operator upon configuration.

There is currently much controversy about whether host configuration languages should be imperative [24, 34, 35] or declarative [1, 4, 5, 6, 7, 10, 22, 23, 26, 27, 33]. A subset of a language is "imperative" when it describes procedure or process: "what should be done" as an interpretable set of instructions. A subset of a language is "declarative" when it describes "what the result should be" without specifying the method or procedure with which this result is accomplished. For example, saying "the car must be blue" is a declarative statement, while "paint the car blue" is an imperative procedure for assuring the truth of the declarative statement. A particular language can exhibit both properties, specifying some things imperatively and others declaratively.

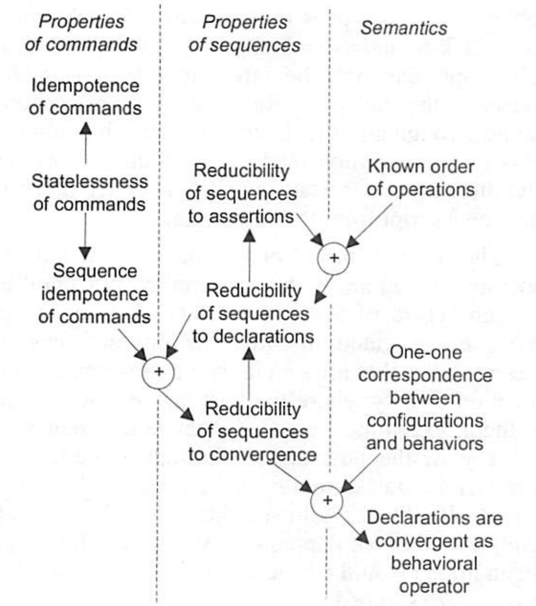


Figure 4: Map of theoretical concepts and their relationships, where arrows indicate logical implication.

Tools support and encourage either imperative or declarative thinking. Proponents of the "imperative" tools point out that specifications for these tools are close to the way a human would manually configure a system, while converting human instructions to declarative language requires some reverse-engineering [24]. Proponents of the "declarative" tools point out that mechanism is not important; one should specify results, not mechanism, and specifying anything more limits flexible response of the configuration management tool to changing requirements [1, 17]. In practice, a majority of seemingly declarative languages for configuration management allow the direct execution of imperative code as an option when declarative mechanisms fail to be expressive enough [4, 5, 6, 7, 11, 13, 22, 23, 33].

While imperative and declarative mechanisms both work well for creating an initial configuration, neither imperative nor declarative mechanisms proves sufficient to implement a closure as described in [17]. In both paradigms, there are serious problems in dealing with *changes in intent over time*. Imperative management mechanisms suffer from script complexity; it is difficult to make changes in these "build scripts" without mistakes [13]. Likewise, undisciplined changes to declarative specifications can lead to unintentional heterogeneity within large networks [17].

Shortcomings of Imperative Scripts

Imperative mechanisms substitute order and reproducibility for understanding of internal dependencies. The script that builds a host is constructed through intensive validation of its behavioral effects, but what the script actually does typically remains poorly understood. This leads to a change control problem as the script is reused over a long lifecycle. Due to lack of understanding of internal dependencies, such scripts can only be safely modified by adding stanzas to the end [24]. Re-imaging a host requires cycling through all of its historical states, including all errors in configuration made by previous scripts. The only alternative is to start over from scratch and validate a new script from the beginning.

The reason that "order matters" within the imperative paradigm is that the implicit preconditions for each stanza of a script are that all prior stanzas have been executed in order. ISConf enforces this order for hosts that miss an update by resuming stanza execution exactly where the host last stopped executing them, executing missed stanzas before new ones. In this way, the host passes through a sequence of reproducible states, so that a final desirable state is assured. If ISConf allowed hosts to skip stanzas, scripts would break unpredictably, because the scripts' preconditions would not be assured on hosts on which stanzas were skipped.

Shortcomings of Declarative Languages

Further, careless use of declarative tools can lead to exactly the kind of unpredictable heterogeneity that the imperative tools like ISConf are designed to avoid. Consider configuration elements A, B, and C with initial values $A = a, B = b, C = c$ and configuration files (declarations) d_A, d_B, d_C. Suppose that

1. d_A sets $A = a'$ and leaves all else alone.
2. d_B sets $B = b'$ and leaves all else alone.
3. d_C sets $C = c'$ and leaves all else alone.

Suppose that at any time, a distinct subset of hosts is down (unreachable). At the end of applying d_A, d_B, d_C in sequence, there are now eight kinds of hosts in the network:

1. $A = a', B = b', C = c'$: Up during d_A, d_B, d_C.
2. $A = a, B = b', C = c'$: Down during d_A; up during d_B, d_C.
3. $A = a', B = b, C = c'$: Down during d_B; up during d_A, d_C.
4. $A = a, B = b, C = c'$: Down during d_A, d_B; up during d_C.
5. $A = a', B = b', C = c$: Down during d_C; up during d_A, d_B.
6. $A = a, B = b', C = c$: Down during d_A, d_C; up during d_B.
7. $A = a', B = b, C = c$: Down during d_B, d_C; up during d_A.
8. $A = a, B = b, C = c$: Down during d_A, d_B, d_C.

As time goes on, the unintentional heterogeneity gets worse, a factor of two at a time, every time a station is unavailable for an update.

Two Principles

The above observations can be summarized as two related principles of configuration management that apply to any such process:

> **Principle 1** Once one controls or manages a thing, one cannot forget over time that it is controlled or managed.[2]

For example, "forgetting" that A is managed above leads to a heterogeneous population of hosts with two differing values of A [18]. More generally,

> **Principle 2** The discipline with which one changes a declarative configuration file is as important to effective configuration management as the accuracy with which the file expresses intent.

Tools that generate the whole configuration for each host each time [1, 2, 10, 22, 23, 26, 27, 33] neatly avoid this problem, at the cost of being somewhat limited in scope and unable to handle large changes such as software subsystem installation and removal.

Transactional Languages

With these goals in mind, we defined a new kind of configuration language that has both imperative and declarative aspects.

> **Definition 1** A *transactional* configuration language is one in which configuration is expressed as a sequence of atomic (indivisible) changes in behavior from a given and known base state.

For purposes of analysis, a transactional language has at least two basic primitives, assert and retract.[3] The command

```
assert {behavior}
```

causes a behavior to be exhibited, while

```
retract {behavior}
```

causes the behavior to become absent. This choice of primitives is arbitrary but allows us to discuss several effects of transactional language easily. The transactions might as well be SQL queries into databases or even XQUERYs into XML.

A transactional language has somewhat of an imperative quality to it, because order sometimes matters, e.g., the order of assert and retract for the same behavior determines whether that behavior is present. The key to our argument and work is that it is also possible – by design – to give the transactional language a declarative flavor as well.

Reducibility to Assertions

At present, our language has no constraints; most any kind of assert and retract statements are allowed. Our next job is to make it possible to simplify complex command sequences.

[2]"Be careful what you command, my son. A command, once given, must be repeated forever." – Duke Leto Atreides, Frank Herbert's *Dune*

[3]Any resemblance to the primitives with the same names in the programming language Prolog is purely intentional.

Definition 2 A transactional language $\mathcal{L}$ containing only assert and retract statements is *reducible to assertions* if for any sequence of assert and retract statements, there is a subsequence of assert statements alone that has the exact same behavioral effect.

Reducibility means that an assertion cannot apply "halfway." If there is a state in which a retraction cancels part of an assertion, then the assertion is "half right." In a reducible language, retractions cancel assertions either fully or not at all.

As an example of a non-reducible language, suppose that directory indexing is turned on by default and one must manually retract the behavior after asserting the contents of the directory. To make this language reducible to assertions, indexing instead must be off by default.

Reducibility has a subtle but important effect upon language. If a language is reducible, the results of any set of transactions can be expressed with "positive language": what should happen, without mention of what should not happen. Since retractions are order-dependent, but assertions typically are not, making a language reducible to assertions has the primary effect that one can express behavioral outcomes in largely order-independent fashion.

Reducibility to Declarations

The ability to eliminate retractions from a language is just one kind of reducibility:

Definition 3 Let $\mathcal{L}$ be a transactional language and let

$$\mathcal{P} = \{(s_a, s_b) \mid s_a, s_b \in \mathcal{L}\}$$

be a partial order on elements of $\mathcal{L}$, where $(s_a, s_b) \in \mathcal{P}$ exactly when s_a must precede s_b. Then $\mathcal{L}$ is *reducible to declarations* if for every sequence of transactions $(t_1, \ldots, t_n)$, there is a subset $\mathcal{D} = \{d_1, \ldots, d_k\} \subset \{t_1, \ldots, t_n\}$ of the set of transactions (where duplicates are eliminated), where for every total ordering $(e_1, \ldots, e_k)$ of $\mathcal{D}$ consistent with the partial order $\mathcal{P}$, the behavioral result of applying the sequence $(e_1, \ldots, e_k)$ is the same as that of applying the sequence $(t_1, \ldots, t_n)$.

This is a complex and perhaps overly wordy way of expressing a relatively simple idea. A transactional language is reducible to declarations if for every sequence of transactions in the language, there is a subset that does the same thing in any reasonable order in which it is applied. In writing down this subset, "order does not matter" because we already know the partial order $\mathcal{P}$ describing how to appropriately order execution of the particular transactions within the subset.

For example, suppose that we have the following transactions:

```
assert A
assert A.X
assert B
assert B.Y
retract B
```

and the partial order:

```
{ (assert A, assert A.X),
  (assert B, assert B.Y) }
```

meaning that one must create A or B before creating their substructures A.X or B.Y. Suppose that retract B retracts the substructure as well; this is allowed. Then we can write an equivalent set of operations as the set { assert A.X, assert A } where the order of this set is unimportant, because we know that assert A.X must follow assert A from the partial order. In our closure, the order constraints are that one must post the contents of parent directories before posting subdirectories; the effect is identical to that in this example.

Whether a language is "declarative" depends upon what we know about the elements of that language and their sequencing. If we are absolutely sure of the appropriate sequences, the order of writing down the elements does not matter; we can resort them into an appropriate order later. A transactional language is reducible to declarations if we can eliminate conflicts from the sequence of declarations so that order does not matter in the resulting reduced set.

Statelessness

While our task requires operations that are reducible to declarations, this is not quite enough:

Definition 4 A transaction or sequence of transactions p is *idempotent* if repeating p twice in succession has the same effect as executing p once.

In other words, once p is successful, doing p again does nothing. More generally,

Let $\mathcal{L}$ be a set of commands. $\mathcal{A}$ is *stateless* if for any command $p \in \mathcal{L}$ and any sequence $q_1, \ldots, q_n \in \mathcal{L}$, applying p followed by $q_1, \ldots, q_n$ followed by p has the same effect as the sequence $q_1, \ldots, q_n, p$. In other words, the initial execution of p before the sequence does not matter.

Statelessness trivially implies idempotence.

The reason statelessness is important is that it is related to idempotence of sequences:

Definition 5 A set of commands $\mathcal{L}$ is *sequence-idempotent* if for any sequence of commands $p_1, \ldots, p_n$ from $\mathcal{L}$, applying the sequence twice has the same effect as applying it once.

This is important because

Proposition 1 If $\mathcal{L}$ is stateless, then $\mathcal{A}$ is sequence-idempotent, i.e., the set of all sequences of commands taken from $\mathcal{L}$ is idempotent.

The proof of this is contained in [16]. This fact allows us to translate between descriptions of a configuration that are command-based and those that are instead declarative:

> **Definition 6** A language $\mathcal{L}$ is *reducible to convergence* if it is reducible to declarations and the declarations, used upon the closure, are idempotent as a sequence.

In other words, given any sequence $(t_1, \ldots, t_n), t_i \in \mathcal{L}$, there is a subset $d_1, \ldots, d_k$ such that if $(e_1, \ldots, e_k)$ is an ordering of $d_1, \ldots, d_k$ conforming to the partial order $\mathcal{P}$, applying the *sequence* $(e_1, \ldots, e_k)$ twice does nothing different than applying it once. In particular, applying this sequence to the configured closure does nothing at all, while applying it to an unconfigured closure reconstructs the closure's current state.

Thus we have the immediate result that:

> **Proposition 2** If $\mathcal{L}$ is both reducible to declarations and stateless, then $\mathcal{L}$ is reducible to convergence.

This is a trivial corollary to Proposition 1. Note that statelessness is a sufficient but not necessarily required condition for sequence idempotence, so that it is a sufficient but not necessarily required condition for being reducible to convergence.

Finally, we relate this to behavior of the overall closure:

> **Proposition 3** Suppose that there is a one-to-one correspondence between behaviors and configurations, and that $\mathcal{L}$ is a set of transactions that change configuration. Suppose that $\mathcal{L}$ is reducible to convergence, $(t_1, \ldots, t_n)$ is a sequence of operations in $\mathcal{A}$ reducible to the declarations $\{d_1, \ldots, d_k\}$, and $(e_1, \ldots, e_k)$ is one order of $d_1, \ldots, d_k$ compliant with the partial order $\mathcal{P}$. Then the operator $e_1 \cdots e_k$ formed by applying $e_1, \ldots, e_k$ in order is convergent in the sense of [4, 5, 6, 7], i.e., it is idempotent as a sequence and interchangeable with the sequence $t_1 \cdots t_n$ in assuring the same behaviors.

Proof: Given $(t_1, \ldots, t_n)$, we know that $\mathcal{L}$ is reducible to convergence, so that $(e_1, \ldots, e_k)$ exists by definition. We also know that from a behavioral standpoint, applying $e_1 \cdots e_k$ has the same *behavioral* effect as applying $t_1 \cdots t_n$. Since there is a one-to-one map between behavior and configuration, these also therefore assert the exact same configuration. Since $e_1 \cdots e_k$ is idempotent, it will not change that configuration. □

This is a bit tricky, as one must require a correspondence between behavior and configuration. In cases where gratuitous differences exist between configuration and behavior, we can still get into a state where $u_1 \cdots u_k$ is not idempotent while $t_1 \cdots t_n$ is. For example, consider the transactions:

```
t1 = { A:=B }
t2 = { B:=4 }
```

and suppose that the value of A affects behavior but the value of B does not. Due to this, $t_1 t_2$ is reducible to t_1, but applying the sequence $t_1 t_2 t_1$ results in differing behavior than applying $(t_1 t_2)$ alone. There are two solutions to this: either disallow use of non-behavioral values in transactions [18], or limit one's self to a definition of configuration in which non-behavioral values do not appear.

Impact of Statelessness and Reducibility

The purport of the above mathematical discussion is that if the commands in a language are stateless, then reducibility to convergence implies reproducibility of effect, i.e., the reduction suffices as a declaration of state. There are several benefits of having a configuration language with these properties:

1. At all times, it is possible to express the effect of a sequence of commands in the same language that the commands use themselves. This eliminates the problem of "semantic distance" [13] in which the language used to declare state differs substantively from the language used to assure it.
2. This effect is expressed in terms of positive and non-conflicting assertions.
3. In a recovery situation, these assertions suffice as commands to reproduce current state.

These are desirable properties for any language, but statelessness would seem a very strong condition upon a language. What kinds of languages have we eliminated from consideration?

A stateless language is simply one in which all assertions are made with absolute (constant) values for parameters (or, at least, parameters that can be converted unambiguously to absolute form, such as relative pathnames). A stateless language cannot allow incrementing or decrementing a configuration parameter, or base one parameter's value upon that of another. This is a stateful (and non-idempotent) change by nature (i.e., $pp \neq p$).

More subtle, the identity of the parameter that gets set by an operation p cannot be a function of some other setting. Suppose we have parameters A, B, C, and that the operation p sets B to 1 if A is 0, and C to 1 otherwise. Suppose that A is initially 0 and the operation q is $A := 1$. Then pqp has a different effect $(A = 1, B = 1, C = 1)$ than qp $(A = 1, B = 0, C = 1)$, violating statelessness.

Stateless Transactions and XML

It would seem that reducibility to assertions and statelessness impose rather extreme limits on what a command language can do. An immediate question about stateless transactions is whether one can create a set of representable (reducible and stateless) transactions that can maintain any kind of configuration file. Most configuration files are hierarchical in structure, and any reasonably consistent hierarchical structure can be expressed by an XML document type definition

(DTD), so it suffices to show that one can correctly maintain the contents of XML files via stateless and reducible transactions.

The allowable forms of an XML document are described by its Document Type Definition (DTD). The DTD describes the allowable contents of each kind of XML node by a *DTD rule*. This rule expresses the structure of allowable content for the node as a regular expression in which tokens are node labels. There are three constructions one can use to make a DTD rule:

1. Sequencing: the expression "A,B,C" matches a sequence of nodes: a node named A followed by a node named B followed by a node named C.
2. Alternation: "A|B|C" matches exactly one of A, B, or C.
3. Repetition: "A*" matches zero or more copies of a node named A in sequence. This is the "Kleene star" operator.

More complex specifications are regular expressions containing the above operators, e.g., one can say that a node named A contains either a node named B or a sequence of nodes named C followed by a node named D by describing A's content via the regular expression pattern "B|(C*,D)". This is described in a DTD by the rule

```
<!ELEMENT A (B|(C*,D))>
```

This rule allows XML such as

```
<A>
<B>...</B>
</A>
```

and

```
<A>
<C>...</C>
<C>...</C>
<D>...</D>
</A>
```

but disallows XML such as

```
<A>
<C>...</C>
</A>
```

(because C cannot appear alone inside A in the above rule). Here ... in the text represents content conforming to (yet to be described) rules for the content of C and D. Other DTD constructions, including + for "one or more instances," are expressible using these constructions: A+ is just "A,A*".

Without loss of generality and to ease notation, we do not consider element ATTRIBUTE declarations in XML. These are easily modeled as subordinate elements of the element to which they apply.

Our initial try at defining XML transactions will be based upon the XPATH language for identifying nodesets within XML files. Every XML file contains nodes that contain other nodes as content. In the file

```
<foo>
  <bar>
    <goo>1</goo>
    <cat>3</cat>
    <goo>4</goo>
  </bar>
</foo>
```

there are five nodes, including one foo, one bar, two goos, and one cat. A *nodeset* within an XML file is a set of nodes within the file having common attributes. XPATH is a language for identifying nodesets, using a notation similar to that used to identify files within a filesystem. For example, the XPATH /foo/bar refers to all nodes named bar within a top-level node named foo, while /foo/bar[2] refers to the second such node named bar in sequence. The special component * refers to a component with any name; /foo/*[5] refers to the fifth component of the content of foo with any name. While this simple subset of XPATH suffices for our discussion, there are many other options too numerous to cover here. All that we need to remember for now is that an XPATH determines a set of nodes within the document for which the assertion controls resulting content. This set of nodes is uniquely determined by the current content of an XML document and an XPATH, and may be empty.

The general form of an XML transaction is:

```
assert <nodeset> <content>
```

where <nodeset> specifies a set of nodes in the file to transform (in XPATH notation) and <content> is XML content that should replace any existing content in the nodeset. <content> can be empty, in which case the assertion has the effect of retracting content from the node. The assertion succeeds if the requested transaction is possible (the nodeset defined by the XPATH <nodeset> is non-empty) and the resulting transformed XML document conforms to the document's DTD; otherwise it fails and does nothing to the document at all. Because DTDs describe the content allowable for each node, and because each assertion provides that content, either all replacements are legal or all replacements fail, together.

For example, in the file

```
<foo>
  <bar>
    <goo>1</goo>
    <cat>3</cat>
    <goo>4</goo>
  </bar>
</foo>
```

performing the command

```
assert /foo/bar/goo[2] <ho>5</ho>
```

would result in the document

```
<foo>
  <bar>
    <goo>1</goo>
    <cat>3</cat>
    <goo><ho>5</ho></goo>
```

```
  </bar>
</foo>
```

(provided that the resulting document is acceptable according to the DTD for the document). The XPATH /foo/bar/goo[2] matches the second instance of goo inside an instance of bar inside an instance of foo. The XPATH /foo/bar/goo would match *both* instances, so that the assertion

```
assert /foo/bar/goo <ho>5</ho>
```

will produce the document

```
<foo>
  <bar>
    <goo><ho>5</ho></goo>
    <cat>3</cat>
    <goo><ho>5</ho></goo>
  </bar>
</foo>
```

Since one can re-assert the contents of the top-level node (via assert /), every state of the XML file is reachable via such assertions. Also, such assertions are idempotent; either an assertion succeeds (if its *total* effect conforms to the document's DTD) or it does not succeed and does nothing to the document. If it does or does not succeed, repeating it immediately has the exact same effect (because the whole assertion is variable-free).

It is a bit more difficult to see that

Proposition 4 The set of all possible assertions $\mathcal{A}$ of the form

```
assert <nodeset> <content>
```

is stateless and reducible to assertions.

Proof: Consider a transaction $T \in \mathcal{A}$ and a sequence S of transactions in $\mathcal{A}$. Note that all content of T is constant; there are no variables that can change state within the transaction itself. Note also that no transaction in $\mathcal{A}$ can change the number of elements in the content of an element unless it also asserts all of the content of that element.

Consider what happens when one applies TST in sequence. Either the nodeset determined by the XPATH in T changes or it stays the same. If it stays the same, then T has the same effect by definition, so that the first T need not be executed and T is stateless. If the nodeset changes, however, it must have changed as a result of assertions that change the whole content of nodes. These assertions must override the prior values of T whenever they affect its nodeset. So in either case, T is stateless. As T and S were arbitrary, the whole language $\mathcal{A}$ is stateless.

$\mathcal{A}$ is trivially reducible to assertions, as it has no retract statements at all! □

Statelessness and Semantics

So far, we have a very awkward system for editing XML. It would be convenient to add two new primitives

```
add <nodeset> <content>
```

and

```
subtract <nodeset> <content>
```

that augment and remove sequenced content from a node. add puts new content into a sequence, while subtract removes matching content from a sequence. The success of both operations is again dependent upon conformance between the result and the document's DTD.

Without further constraints, we no longer have statelessness. To have statelessness, we must also have idempotence, but the add primitive is not even idempotent; adding something twice results in two entries for the item instead of one. For example, consider the XML document

```
<foo>
  <bar>
    <goo>10</goo>
    <goo>16</goo>
  </bar>
</foo>
```

and the effect of two statements:

```
add /foo/bar <goo>20</goo>
add /foo/bar <goo>20</goo>
```

After these statements (and with no further constraints) the resulting document would contain:

```
<foo>
  <bar>
    <goo>10</goo>
    <goo>16</goo>
    <goo>20</goo>
    <goo>20</goo>
  </bar>
</foo>
```

instead of

```
<foo>
  <bar>
    <goo>10</goo>
    <goo>16</goo>
    <goo>20</goo>
  </bar>
</foo>
```

(which would be the required result for a stateless add operation).

The DTD for an XML document is syntactic; it describes what can be written but not what the written text means. What is missing from our model is a notion of *semantics* that would tell us when two configurations are equivalent. A model of semantics, in turn, allows one to understand what add and subtract should do in order to remain stateless.

Note that idempotence and statelessness are not properties of what operations do, but of *what the results mean*. If operations act on a configuration file to produce the same contents, it is rather obvious that they result in the same behavior. However, sequences of operations that produce differing configuration files may produce the same behavior, e.g., if the differences in configuration do not produce differences in behavior.

In adding information to a sequence, one must ask several questions. Does order of the sequence matter or not? Do duplicates matter, or does the first or last instance of a duplicate override the others? What constitutes a duplicate entry? These are *semantic* questions that go beyond the simple *syntax* described by a DTD.

Preserving statelessness in using substructure addition and subtraction is a matter of both understanding semantics and limiting operations to fit. If members of a sequence are pure declarations, so that order does not matter and duplicates are ignored, then add and subtract should behave accordingly. Trying to add a duplicate should have no effect.

More subtle, the semantic definition of a duplicate often involves the notion of a unique key. For example, suppose that in Apache we are defining access rights to directories. Obviously, the directory to which we are defining access constitutes a unique key; we should not be able to define two different ideas of protection for one directory.

This means that we need several context-sensitive notions of what add and subtract should do.

1. If order of assertions matters and cannot be inferred from assertion content, the game is over. Language is imperative and statelessness is impossible.
2. If order of assertions does not matter and duplicates can occur, then add cannot be stateless, because it is not even idempotent.
3. If order of assertions does not matter and duplicates should not occur, the add command should have the form

   ```
   add <nodeset> <key> <content>
   ```

   where <nodeset> determines a set of nodes on which to operate, <key> is an XPATH *relative* to each node that determines a key that should be unique, and <content> is content to add. For example, given the XML

   ```
   <foo>
     <bar>
       <goo>10</goo>
       <goo>16</goo>
     </bar>
   </foo>
   ```

   the command

   ```
   add /foo/bar goo <goo>10</goo>
   ```

   does nothing at all, because 10 is already a key, while

   ```
   add /foo/bar goo <goo>20</goo>
   ```

   results in the document

   ```
   <foo>
     <bar>
       <goo>10</goo>
       <goo>16</goo>
       <goo>20</goo>
     </bar>
   </foo>
   ```

   The extra goo in the assertion indicates that the *content* of goo is the key that should be unique.

The moral of this section is that unless the XML being created is truly a declaration (i.e., order does not matter and duplicates should not exist), statelessness is impossible in the transactional language that updates it.

Application to Configuration Management

The impact of this mathematical theory upon the general problem of configuration management is subtle but inescapable. So far, we have largely ignored the problem of change management in host configuration management. Generative tools (that create an entire configuration from templates) largely avoid the issue of change management by erasing everything and starting over each time. However, these tools are relatively limited in scope, as they cannot handle major changes such as package management: installation and removal of software subsystems. Convergent tools allow one to become sloppy and forget that a component is managed, while imperative tools deal poorly with undoing changes.

In the sub-problem of managing the configuration of a web server, change management is a central concern, so that we must adjust our practice and language to ease that task. The result, however, is that we created a framework for change management that applies to the more general problem of network configuration management. In fact, we have created an "assembly language" that is the lowest level of a new strategy for configuration management.

At its core, every configuration can be described in terms of a set of assertions that are true at a given time. In the simplest case, each assertion assigns values to one or more "configuration parameters." Since configuration values are always specified as absolute quantities in assertions, such assertions are naturally stateless. Most configuration management tools are driven by configuration files containing only stateless assertions of this kind.

By viewing the assertions in such a configuration file as commands to be executed, the only thing we have added in our model is a concept of retraction of assertions. There is a constraint model that describes which assertions conflict with which others, and a rule that keeps current values consistent with one another, e.g., for our web server, we know that asserting a new index file for a directory is going to override the old index declaration. If we retract a virtual server, then all parameters for that server no longer exist.

If the command language is reducible to assertions, no matter what incremental changes we make to the overall configuration through further assertions, the results remain precisely expressible as a set of assertions. Further, the assertions are sequence-idempotent, so that repeating them has the same effect as doing them once. Thus the list of valid assertions is a

good substitute for the policy file found in many configuration management systems.

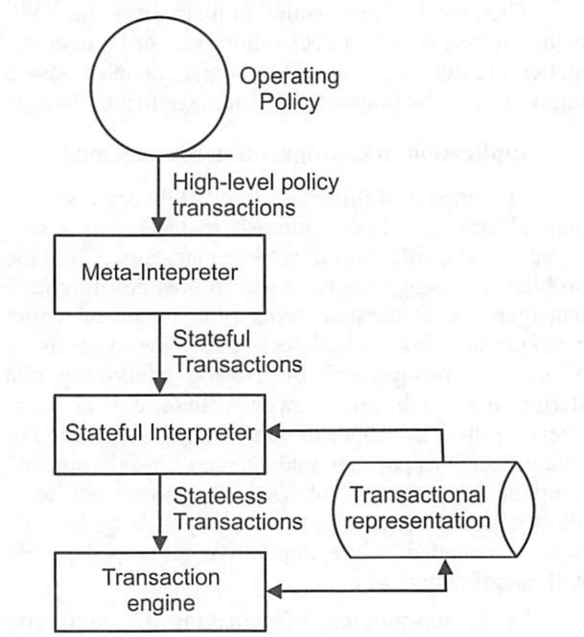


Figure 5: A new model of configuration management, in which low-level statements exhibit statelessness while upper levels encapsulate stateful behavior.

This gives rise to a new model of configuration management (see Figure 5). At the core, a transaction engine interprets a stateless language. This engine interprets stateless commands to effect requested changes in overall configuration. This engine is responsible for maintaining the list of valid assertions and retracting conflicting assertions. This engine deals with stateless commands only.

At the next level, stateful commands are translated into stateless commands for ease of use. While the user thinks in relative terms ("more space"), the transactional system must think in terms of absolute requirements ("2 MB"). A simple memory mechanism here makes the translation between relative and absolute units.

At the third level, meta-commands describing overall intent are translated into the assertions that create that intent. "Become a web server" is translated into the various assertions that cause that to happen.

This rather strange way of accomplishing configuration management has a few rather obvious advantages:

1. The whole history of the configuration of the machine is contained in one transaction stream.
2. At any time, there is a deterministic procedure that can determine what configuration is in effect at that time.
3. Storing the stream and its changes allows one to roll back time, by replaying the stream or selectively retracting the newest assertions, backwards.
4. One can specify changes as incremental operations upon a pre-existing structure.

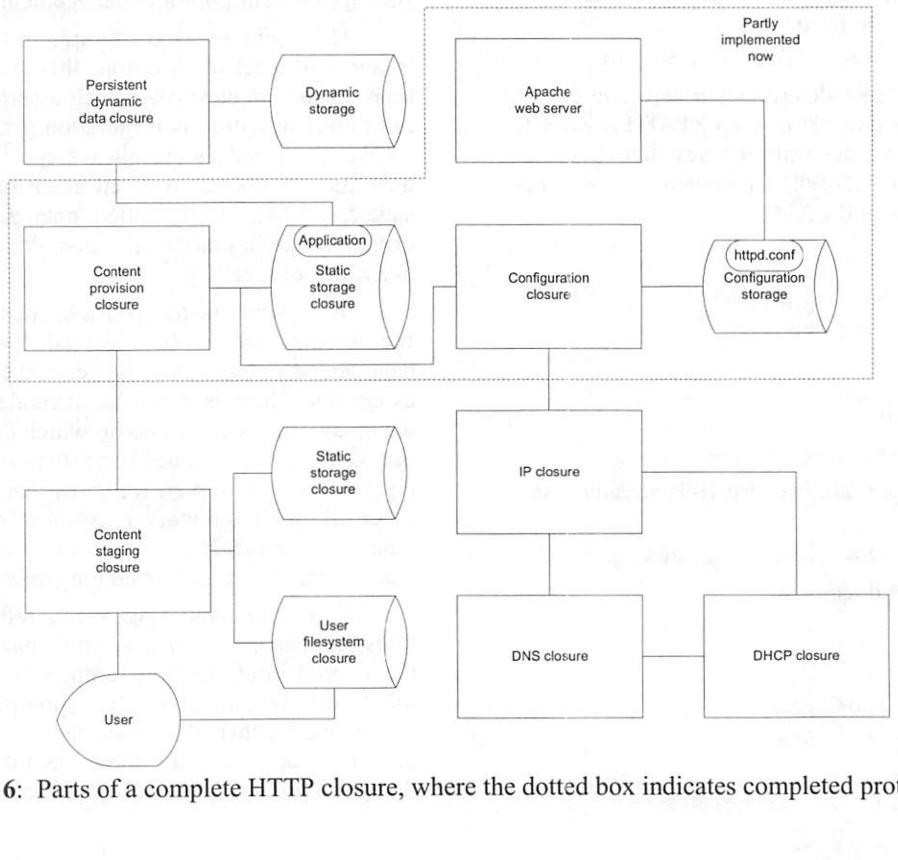


Figure 6: Parts of a complete HTTP closure, where the dotted box indicates completed prototypes.

5. Changes can come from multiple sources (e.g., different administrators or groups) and will be disambiguated at implementation time.
6. At any time, there is a coherent picture of the assertions currently in effect.

These observations are *not* true in general of current convergent administrative tools, including Cfengine. In Cfengine, changes are specified by editing a monolithic file. There is no easy way to undo a configuration step. It is difficult for multiple people to collaborate on a single configuration without conflicts. In order to implement such a mechanism, Cfengine would have to have the ability to return a file to the state before any edits have been applied.

This proposal needs much study before we implement such a language, but is clearly implied by our study of HTTP.

Conclusions

Work on this project has been a long road of discovery. Ad-hoc creation of a closure language – based upon a configuration language – led to much initial confusion. The exact things that make a configuration file easy to read make a command language confusing and difficult to use. Disambiguating that language required a mathematical approach: stateless commands removed the quandaries posed by stateful syntax. The result proves that for a limited problem domain, closures exist.

But we are a very long way from coming to a complete closure. A complete solution seems to have more parts than we could have imagined initially(Figure 6).

1. To assure idempotence of operations, we need content staging, and an independent repository for staged data. There should be two content hierarchies, one for actual provision and the other a cached copy that allows restoring the provided data, e.g., after a crash.
2. It is impossible to deal with dynamic content written into the web hierarchy by traditional means. This must be handled by some kind of dynamic storage closure (that may be a database, or perhaps something else).
3. We had much difficulty verifying that a declared virtual server would work properly according to information in DNS and DHCP. We need the ability to converse with and negotiate with DNS and DHCP closures in order to determine whether declared virtual servers will work properly.

As well, the actual configuration closure should have several parts that are not currently present(Figure 7). A module management subsystem should allow dynamic selection of modules, while a constraint engine disallows impractical choices. Likewise, a virtual server management subsystem should disallow virtual server configurations that cannot work, e.g., declaring more than one ssl server for a single IP address.

We also acknowledge that the end product may not be a single http service closure, but several differing ones for different applications. Making the language simple enough to use in one application may preclude its use in another. For example, a closure whose language is simple enough for use by untrained

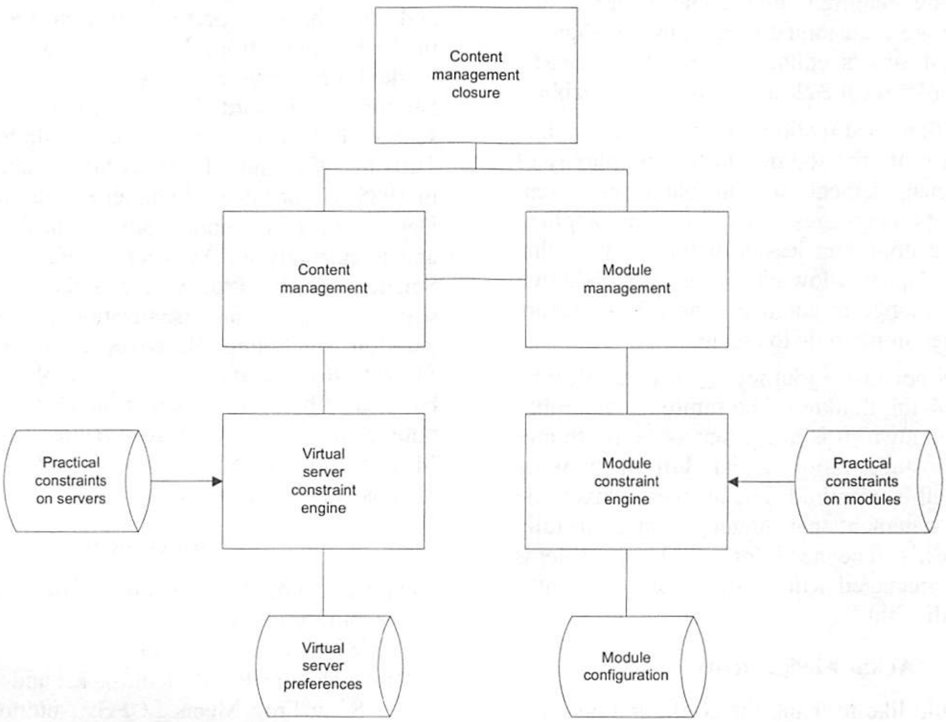


Figure 7: Parts of the configuration management closure in Figure 6.

people may not be expressive enough to be appropriate for experts.

This is only the beginning of our journey, and there are several caveats for those who would follow this path.

1. First, beware of seemingly stateless languages whose *environment* is stateful. In our language, the act of copying files is stateless, but the files themselves can change between copies. So the copying commands are not truly idempotent.
2. Second, beware of implementing the top level of a closure before the lower levels. We cannot really *assure* behavior, because our closure is not built on top of a foundation of lower-level closures. Assurance and trust must rise from the bottom levels rather than being imposed from above. This is the obvious way to settle quandaries such as how to validate that virtual servers will receive requests, etc.
3. Third, beware of making a closure that works around limitations that should rightly be there with good reason. We chose not to allow CGI scripts to create dynamic content within the closure. This seems a limitation, but actually reflects best practices for web content creation. If CGIs "should" be using databases, why should we allow them not to? Ideally these CGIs should be communicating with "data persistence closures," otherwise known as database management systems!
4. Creating a closure that reacts predictably to configuration commands requires discipline in creating the command set. But many more disciplines are required, and some features to which we are accustomed in existing paradigms – like CGI scripts editing server files – must cease in order for the closure to become reliable.

In the final estimation, it remains unclear whether closures are the solution to the complexity of configuration management, and unclear whether reasonable stateless languages exist for other applications. The most important lesson of this study is that practice must adapt to allow closure to exist. Without a fundamental change in language, the HTTP closure would have been impossible to create.

Ours was not just a journey of software design, but also of evolving thinking. The future of that thinking remains unknown. It is likely that as we try to create practical closures, more radical shifts in thinking and practice will be required. The answer seems to lie in the simple statement that language must carefully conform to needs. The need for simple subsystems that are easily managed will no doubt lead to "paths where no one thought."

Acknowledgements

Alva would like to thank the configuration management community for their continuing support; notably Mark Burgess, John Sechrest, and Paul Anderson. Without their encouragement, the difficult parts of this project would have seemed impossible. Thanks also to long-time colleague David Krumme who was the initial sounding board for many of these ideas.

Steven would like to thank his family and friends for their support, including Ken D'Ambrosio for being a sounding board and an invaluable resource in refining and codifying many concepts into the final form. Thanks to Lara Ullman, Micha Rieser, and John Knoerzer for being there as a sympathetic ear, and forcing a strung out graduate student to take a break when one was truly needed. Thanks must also go out to Lynne Baer for her patience and unwavering faith in me. Lastly, he would like to thank everyone reading this paper, this being my first foray into publication; hopefully you have enjoyed reading it.

Author Information

Steven Schwartzberg has been working in the system administration field for seven years in various capacities ranging from a techie at a small ISP to managing a team at a small technology start-up. He recently received his Masters of Science from Tufts University in the field of Computer Science. He is currently working as a system engineer at BBN Technologies in the Decision and Security Technologies Division, and can be contacted via electronic mail as sschwart@bbn.com .

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from MIT in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Computer Science at Tufts. Prof. Couch is the author of several software systems for visualization and system administration, including Seecube(1987), Seeplex(1990), Slink(1996), Distr(1997), and Babble(2000). He can be reached by surface mail at the Department of Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@cs.tufts.edu .

References

[1] Anderson, P., "Towards a High-Level Machine Configuration System," *Proc. LISA VIII*, USENIX Assoc., 1994.

[2] Anderson, P., P. Goldsack, and J. Patterson, "SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control," *Proc.*

LISA XVII, USENIX Assoc., San Diego, CA, 2003.
[3] Bohlman, E., "Parsing XML, Part 1," http://www.perlmonth.com/perlmonth/issue4/perl_xml.html .
[4] Burgess, M., "A Site Configuration Engine," *Computing Systems*, Num. 8, 1995.
[5] Burgess, M. and R. Ralston, "Distributed Resource Administration Using Cfengine," *Software: Practice and Experience*, Num. 27, 1997.
[6] Burgess, M., "Computer Immunology," *Proc. LISA XII*, USENIX Assoc., 1998.
[7] Burgess, M., "Theoretical System Administration," *Proc. LISA XIV*, USENIX Assoc., 2000.
[8] Cameron, Jamie, "Webmin," http://www.webmin.com/index.html .
[9] Cameron, Jamie, "Swell Technology Virtualmin Virtual Hosting Management," http://www.swelltech.com/virtualmin/ .
[10] Cons, Lionel and Piotr Poznanski, "Pan: A High-Level Configuration Language," *Proc. LISA XVI*, USENIX Assoc., 2002.
[11] Couch, A., "Chaos Out of Order: A Simple, Scalable File Distribution Facility for 'Intentionally Heterogeneous' Networks," *Proc. LISA XI*, USENIX Assoc., 1997.
[12] Couch, A., and M. Gilfix, "It's Elementary, Dear Watson: Applying Logic Programming to Convergent System Management Processes," *Proc. Lisa XIII*, USENIX Assoc., 1999.
[13] Couch, Alva, "An Expectant Chat About Script Maturity," *Proc. LISA XIV*, USENIX Assoc., 2000.
[14] Couch, Alva, and Noah Daniels, "The Maelstrom: Network Service Debugging via 'Ineffective Procedures'," *Proc. LISA XV*, USENIX Assoc., 2001.
[15] Couch, A., and Y. Sun, "Global Impact Analysis of Dynamic Library Dependencies," *Proc. LISA XV*, USENIX Assoc., 2001.
[16] Couch, A., and Y. Sun, "On the Algebraic Structure of Convergence," *Proc. DSOM'03*, Elsevier, Oct 2003.
[17] Couch, A., J. Hart, E. Idhaw, and D. Kallas, "Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management," *Proc. LISA XVII*, USENIX Assoc., 2003.
[18] Couch, A. and Y. Sun, "On Observed Reproducibility in Network Configuration Management," to appear in the *Science of Computer Programming* special issue on Network and System Administration, Elsevier, Inc, 2004.
[19] Gélinas, Jacques, "Linuxconf Home," http://www.solucorp.qc.ca/linuxconf/ .
[20] Giridharagopal, Deepak, "DryDock: A Document Firewall," *Proc. LISA XVII*, USENIX Assoc., 2003.
[21] Hart, J. and J. D'Amelia, "An Analysis of RPM Validation Drift," *Proc. LISA XVI*, USENIX Assoc., 2002.
[22] Holgate, M. and W. Partain, "The Arusha Project: A framework for collaborative Unix System Administration," *Proc. LISA XV*, USENIX Assoc., 2001.
[23] Holgate, M., W. Partain, et al., "The Arusha Project Web Site," http://ark.sourceforge.net .
[24] Kanies, L., "Isconf: Theory, Practice, and Beyond," *Proc. LISA XVII*, USENIX Assoc., 2003.
[25] Sandnes, Frode Eika, "Scheduling Partially Ordered Events in a Randomised Framework – Empirical Results and Implications for Automatic Configuration Management," *Proc. LISA XV*, USENIX Assoc., 2001.
[26] Finke, Jon, "An Improved Approach for Generating Configuration Files from a Database," *Proc. LISA XIV*, USENIX Assoc., 2000.
[27] Finke, Jon, "Generating Configuration Files: The Director's Cut," *Proc. LISA XVII*, USENIX Assoc., 2003.
[28] Harold, E., and S. Means, "XML in a Nutshell, Seconnd Edition," O'Reilly, Inc, 2002.
[29] Logan, Mark, Matthias Felleisen, and David Blank-Edelman, "Environmental Acquisition in Network Management," *Proc. LISA XVI*, USENIX Assoc., 2002.
[30] Oetiker, T., "TemplateTree II: the Post-Installation Setup Tool," *Proc. LISA XV*, USENIX Assoc., 2001.
[31] Patterson, J., "A Simple Model of the Cost of Downtime," *Proc. LISA XVI*, USENIX Assoc., 2002.
[32] Daniel López Ridruejo, "Comanche Downloads," http://www.comanche.org/downloads/ .
[33] Roth, M. D., "Preventing Wheel Reinvention: The Psgconf System Configuration Framework," *Proc. LISA XVII*, USENIX Assoc., 2003.
[34] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," *Proc LISA XII*, USENIX Assoc., 1998.
[35] Traugott, Steve and Lance Brown, "Why Order Matters: Turing Equivalence in Automated Systems Administration" *Proc. LISA XVI*, USENIX Assoc., 2002.
[36] Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, J. Norris, M. S. Lam, and M. Rosenblum, "Virtual Appliances for Deploying and Maintaining Software," *Proc. LISA XVII*, USENIX Assoc., 2003.
[37] Wang, Yi-Min, Chad Verbowski, John Dunagan, Yu Chen, Chun Yuan, Helen J. Wang, and Zheng Zhang, "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. LISA XVII*, USENIX Assoc., 2003.

[38] Ginger Alliance, "Charlie – XML Application Framework System," http://www.gingerall.com/charlie/ga/xml/p_ch.xml .
[39] The Gallery Team, "Gallery – Your Photos on Your Website," http://gallery.sourceforge.net .
[40] XML Working Group, "XML Schema Specification," http://www.w3c.org .

Meta Change Queue: Tracking Changes to People, Places, and Things

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Managing information flow between different parts of the enterprise information infrastructure can be a daunting task. We have grown too large to send the complete lists around anymore, instead we need to send just the changes of interest to the systems that want them. In addition, we wanted to eliminate "sneaker net" and have the systems communicate directly without human intervention. Some of our applications required real time updates, and for all cases, we needed to respect the "business rules" of the destination systems when entering information. This paper describes a general method for propagating changes of information while respecting the needs of the target systems.

Introduction

At LISA 2002, I presented a paper *Embracing and Extending Windows 2000* [4] that described how we kept our Windows 2000 environment, as well as our LDAP directory services synchronised with our Unix account space. These feeds quickly grew to carry more than just Unix account information to include directory and other status information. Well, we were a victim of our own success. Other systems needed access to the same or similar change feeds, and other data streams were becoming available, and a more general architecture was needed. In addition, we found that we had to interface with vendor supplied systems and it became important to provide a clear demarcation between our systems and the vendor's systems and provide a clear place to implement their business rules with our data.

At LISA 96 in Chicago, I gave an invited talk *Manage People, not Userids* that demonstrated the importance of managing the more general information about people, and from that, managing their computer accounts. In addition, in a paper at the same conference, (*White Pages as a Problem in Systems Administration* [3]), I again showed how tools for systems administration could benefit other areas and that many areas for code and tool re-use exist. As our friends in the JAVA community (and other object oriented languages) are fond of telling us, solve the problem once, and re-use the solution to solve other problems. Thus, we wanted a general mechanism to move different types of changes to different systems.

At our site, many of our systems[1] are vendor supplied packages running on an Oracle or other relational database. In addition, we were also feeding information to non relational database systems such as our LDAP directory servers and the Windows 2000 domain controllers. To further complicate matters, we have many different data elements available, and not all systems wanted all data elements, we needed ways to pick and choose which data elements went to which system. We also needed to be able to accommodate different operating schedules and data latency requirements. Some data elements change very slowly (such as adding a new building) where a daily update feed is more than adequate, while other data elements need to move much faster (such as a password change, or email forwarding.) We wanted to retain the low processing costs we achieved in earlier implementations, while making it easier to add new "listeners" to a feed. Lastly, although we wanted changes to propagate quickly, we needed to avoid blocking an operation on one system because a downstream system was not reachable.

[1] Student Records, Human Resources, ID Card, Dining Services, Space Management, Telephone Billing, Help Desk, etc.

Interfaces and Business Rules

The first aspect of this project, is the interface model we use to actually get the changes into the destination system. While many applications have procedures to import a CSV file, these require manual activity and our objective is to fully automate the process. Some applications and systems provide an API that we can call to insert and update records; this is our preferred method. But other systems don't provide that and for at least database based systems, we need to muck about directly in the vendor database tables. We wanted a clear demarkation between our systems, and the interface code that needs to understand how the target system works. For the systems without an API, our approach is to insert the changed records into a import table and have that trigger the appropriate processing. We have used this model as well as the API model successfully.

Assuming that we have some sort of interface, we still need to face the classic system admin issue of pushing in changes from the central server, versus pulling in changes from the client. The answer here is "it depends." In general, I have taken a very pragmatic

approach. For destination systems that do not have "aggressive administration,"[2] I prefer the push model from the central server. This allows me to monitor the connections and updates and become aware of problems (and hopefully resolve them) before the end users. This also allows me to adjust the schedule and timing of updates as needed. For systems with aggressive administration, we can negotiate the "best" approach (more efficient, least work, etc.).

Procedural API

At the heart of the Meta Change Queue package is the Get_Changes routine (Figure 1) which provides all of the changes for the specified listener in order. This is called with the processor (queue) name, and an optional table name within that queue. This will return a record with a number of fields of interest (Table 1). When the record has been processed, the Ack_Change routine is called. This cycle is repeated until the Change_Type field in the record is null. This indicates that there are no more changes that need to be processed.

```
Function Get_Changes(
     Proc_Name in varchar2,
     tname in varchar2)
return rec;
procedure Ack_Change(R in Rec);
```

Figure 1: Get_Changes definition.

When an application is processing a change, it examines the change record, and based on the Tname and subtype (and other fields), determines what record had changed and gets the current value of that record from the database. This is a very important issue to understand, we do not record what the change was, only that something had changed. We need to be able to move the final state for a record, without having to step through intermediate steps. If I change my phone number twice, the only thing that matters is the final number. Other aspects of our systems may maintain history and change logs, but not this one. Here we only indicate that something changed. The application must be able to apply the same change twice without harm, i.e., "set quota to 100" is ok to repeat, "increase quota by 50" is not.

There is another set of routines that given a change record, will return the desired information (directory, status, etc.) to applications that can then update the target system. This model has worked well with our interfaces to LDAP and Active Directory where we have written a program in Java or C#, that gets the queued changes and updates the target. These applications apply all the changes in the queue, acknowledging them as they go. Once it reaches the end of the queue, it sleeps for a short time and looks for changes again. These programs will retry if they loose the network connection and will eventually catch up once they can reconnect. This automatic restart has proven very handy and reliable.

The Get_Changes interface also provides a handy hook for our process monitoring system [5]. The applications that are polling via the Get_Changes routine often just sleep for a short time; maybe a second. Unlike the calls to Get_Changes which puts a very small load on the database, calls to the Mark_Process routine results in a write (or update) to the database, and frequent calls will impact performance and transaction logs. So we typically wrap the call to Mark_Process in code that skips the actual call until at least five minutes has elapsed since the last call. This will still give us good notification when one of these processes dies. We usually catch one that has died every three or four months.

Import Table

Our second interface method is by using an import table to receive records. When a record in inserted into the import table, a database trigger[3] fires which will then process whatever business rules that

[2] An administrative team who is constantly monitoring the system and is able and willing to set up cron jobs or the equivalent.

[3] A database trigger is a stored procedure in the database that will be executed whenever there is an insert, update or delete on a row in a database table [1].

Field	Type	Description
Tname	varchar2(32)	The name of the table that had the change.
Change_Type	varchar2(8)	One of "Insert," "Delete," or "Update."
rrowid	rowid	Oracle row identifier of base table record.
proc_name	varchar2(32)	The processor (or queue) name.
subtype	varchar2(32)	More detailed information about what specifically changed about the target object.
person_id	number	The internal person identifier if the object is defined in the "people" table.
Pkey_String	varchar2(32)	The primary key (identifier) of the object (if not a person) as a character string.
pkey_number	number	The primary key of the object when that is a numeric value (not a character string).
aux_string	varchar2(255)	An optional extra character field to identify the change. Often used for membership changes where two keys are needed.
entry_date	date	The time and date this change was made.

Table 1: Change record definition.

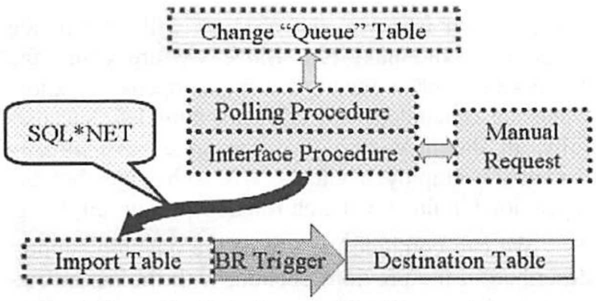


Figure 2: Interface and business rules.

are required. This appears as the bottom row of elements in Figure 2. We have used this successfully with several different vendor applications.[4] In cooperation with the vendor engineers, we define an import table, and then the vendor engineer writes a database trigger that processes each insertion as it happens and makes the appropriate changes in their own tables. This allows us to feed in the changes in a controlled manner and isolates our code from vendor changes. The vendor does need to update their triggers when they make a change. We had originally intended to queue the records in the import table, and then the vendor would have a process that looks for pending records (much like how we did the Meta Change Queue project), but we found it easier to just write the trigger and avoid writing the polling application.

One example of this is seen in Figure 3 which is a database trigger written by a vendor engineer. In this case, for each new entry in the Simon_Person_Import table, it first checks to see if the entry has already been made in the vendor table SA_PERSON, and if not, it inserts the person. If the person is already in the table, it checks to see if the person has a status.[5] If they don't have a status, see if they did, and if so, change it to "Former-" whatever and then update the person's record. The vendor application did not a have a field for the "Status" of a person, and although we could

[4]BEST – ID Card and Access Control, FAMIS – Physical plant trouble ticketing, INSITE – Space Management

[5]Maintaining "status" values for every person is a topic for another paper.

```
CREATE OR REPLACE TRIGGER T_SIMON_PERSON_IMPORT
BEFORE INSERT ON SIMON_PERSON_IMPORT FOR EACH ROW
declare
    Cursor Get_rec (pn number) is
    Select person_number,person_last_name,person_first_name,
           person_text1, person_location, person_memo, rowid
      from SA_Person where person_number = pn;
    R           Get_Rec%RowType;
    new_status  varchar2(48);
begin
    Open Get_Rec(:new.spriden_id);
    Fetch Get_Rec into R;
    if Get_Rec%NotFound        -- No existing record, insert a new one
    then
        new_status := nvl(:new.status,'No Status');
        INSERT INTO SA_PERSON
            (PERSON_ID, ENTERED_DATE, ENTERED_BY, PERSON_NUMBER, PERSON_LAST_NAME,
             PERSON_FIRST_NAME, PERSON_TEXT1, PERSON_LOCATION, PERSON_MEMO)
        VALUES
            (PERSON_ID_SEQ.NEXTVAL, SYSDATE, USER, :new.spriden_id, :new.lastname,
             :new.firstname, orgn, substr(new_status,1,24), :new.title);
    else
        if :new.status is null  -- If no current status, save what they were
       then
            if substr(R.person_location,1,7) = 'Former-'
            then new_status := r.person_location;
            else new_status := 'Former-' || R.Person_Location;
            end if;
        else
           new_status := :new.status;
        end if;
        Update SA_Person
             set Updated_Date = sysdate, Updated_By = user,
                 Person_Last_Name = :new.lastname, Person_First_Name = :new.firstname,
                 Person_Location = substr(new_status,1,24),
                 person_text1 = orgn, person_memo = :new.title
           where rowid = r.rowid;
    end if;
end;
```

Figure 3: Insite Trigger.

have added one to their table (like we did with the Person_Number, we did not want to change all of the display screens, so we took over the Person_Location field, and store and display the status there.

We don't actually care about the contents of the Simon_Person_Import table. Once the trigger fires and completes, all of the work is done. We periodically flush the import table. If there is a problem with the trigger, perhaps some integrity constraint (unique usernames, etc.) is violated, the trigger throws an exception and the insert fails. This exception propagates back to the system attempting the insert and appropriate error handling can take place there.

This approach has the additional advantage of allowing us real time updates for applications that needed it. For example, we have a secure web page that is used by our Human Resources department to mark when a new employee has signed their I9 form (and is now allowed to start work). This web form updates the person's status, and immediately pushes that change to the ID card system. By the time the new employee has made it to the ID desk, they have already been loaded in and can have their ID card photo taken right away. This has made both the HR staff and the ID desk staff happy (HR is happy because they can now control when someone is issued a staff ID card, and the ID Desk staff is happy because they don't need to call HR to verify each new hire.)

Not all changes need to happen in real time. Many changes happen as the result of other automated processes and batch jobs. We have a simple PL/SQL program that uses the Get_Changes routine to find out what has changed for a given queue, and then loads the appropriate records into the import table. If the target system is down, the changes will wait in the queue until the next run. Since we are using the process monitor to ensure that this happens, we know when the scheduled jobs does not complete successfully. In the new employee case, we have already loaded the employee via the HR web page, but the repeat load in the next batch run doesn't hurt anything.

We can combine the use of the queuing support described in the previous section, with the insert trigger based code, to come up with a catch up routine like the one in Figure 4. This simply looks for changes for the 'Insite' queue, and passes them to the Insite system via the Push_Person routine we described earlier. Once we get to the end of the list, we record the fact we finished and terminate. This process is called once a day by a cron job.

Manual Entries

When we bring a new system on line, it is generally empty of our data. Rather than loading it via CSV files or other bulk import tools, we use the Meta Change Queue interface to load them up. In the cases where there is a program calling the Get_Changes routine directly, we simply manually insert records in the queue for that service, and watch what happens. If we like what we see, we write a simple script to load all objects of interest into the queue. From that point on, things run automatically, and the interface has been well tested, as the entire system load has been processed via the new interface. This also makes it easy to reload if we decided to flush and start over.

In Figure 5, we have an example of a PL/SQL script that will select all transfer students from the Fall of 2002, and "refresh" their entries in any listener that

```
procedure Push_Queue(stopcount in number)
is
    R       Meta_Change_Access.Rec;
begin
    loop
        R := Meta_Change_Access.Get_Changes('Insite', Null);
        exit when R.Tname is null;
        Push_Person(R.Person_Id);
        Meta_Change_Access.Ack_Change(R);
    end loop;
    Process_Monitor_Record.Mark_Proc('Insite-Push_People');
end Push_Queue;
```

Figure 4: Push_Queue procedure.

```
declare
    Cursor Lrec is Select Username,Source, owner, unixuid, rowid
      from logins where admit_cohort='TR200209';
begin
    for L in lrec
    loop
        Meta_Change_Rtn.Log_Update('LOGINS',l.rowid, person_id => l.owner,
             pkey_string => l.username, pkey_number => l.unixuid);
    end loop;
end;
```

Figure 5: Manual refresh via queue.

is interested in changes to the LOGINS table.[6] You will note that several of the parameters specified in the call to LOG_UPDATE correspond with fields in the change record (Table 1).

In the cases where we use the import (trigger) table, we generally have written a routine like Insite_Interface.Push_Person (Figure 6) that will look up the appropriate information and do the appropriate insert (via SQL*NET). This routine can be called by hand for testing, and later on via scripts to bulk load the entire population. In Figure 7, we have an example of PL/SQL script that will load all current employees and faculty into the INSITE (space management) system via the Insite_Interface.Push_Person routine. This routine calls routines in the Meta_Change_Data package to get the data elements that are needed, and then inserts that into the import table Simon_Person_Import on the Insite machine using sql*net.

Tables and Listeners

The second aspect of the project, is how we detect changes, queue them, and finally deliver those changes in a timely manner.

Defining Tables

The original concept was to track changes in a particular database table, but in the actual implementation, this proved to be limiting. Instead of looking at the details of the source systems tables, we looked at the data requirements of the destination system. For example, one system might just want general information on a person such as name and status, while another system would want that as well as directory information. Since the transfer model was to give them a complete record of all desired information about a person, a facility to pick and choose what information about a person, was desired. Instead, we defined the table to be the source of the primary key, and added a sub type to indicate what about the base object changed. For example, a telephone number change would be marked as the PEOPLE table and the Telephone sub type. We currently have 16 table and sub type combinations defined (Table 2).

To detect these changes, we set up a database trigger (Figure 8) which records whenever a telephone number is changed. There are similar triggers to handle new telephone numbers (inserts) and deleted telephone numbers. Since this was done with a database trigger, we did not have to change any of the applications that had been previously developed to make changes. It also ensures that we don't miss any changes.

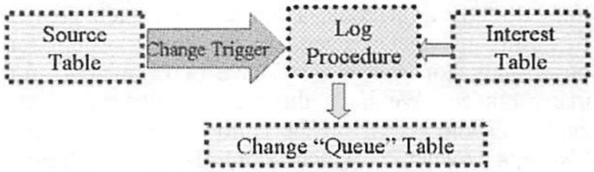


Figure 9: Detecting changes with triggers.

While database triggers can be very handy for integrating existing applications, they can sometimes get complicated. We often have changes to a table that are "housekeeping" in nature. Something in the table changed, but that change is not of interest to any downstream systems. You can with a trigger be more selective about what columns you look for changes in, but that makes the trigger more complex. Triggers are also challenging from a maintenance prospective, as they are sort of split conceptually between the table

[6]The query has been edited for space, but the concept is still valid.

```
Meta_Change_Data.Person(Person_Id, Lname, Fname, Mname,
     PFN, Rin, Iso, DOB, Gender, Ssn, Pidm);
Meta_Change_Data.Person_Department(Person_Id, Department, Division,
     Portfolio, Insite_Name, Orgn_Code;
Meta_Change_Data.Person_Status(Person_Id, Category, ID_Card,);
Meta_Change_Data.Person_Directory(Person_Id, Title, Camp_Add,
     Camp_Phone, Camp_Fax, Mailstop);

Insert into OPS$INSITESYS.SIMON_PERSON_IMPORT@insite
     (Spriden_Id, Lastname, Firstname, Orgn_Code, Status, Title)
Values (Rin, upper(substr(lname,1,24)),upper(substr(nvl(pfn,fname),1,16)),
     Orgn_Code, upper(ID_Card), upper(Title));
```

Figure 6: Insite_Interface.Push_Person.

```
declare
   Cursor Emp_List is
   Select person_id,spriden_id,lastname
     from people
     where id_card_status in ('Employee','Faculty');
begin
    for R in Emp_List loop
        Insite_Interface.Push_Person(R.person_Id);
    end loop;
end;
```

Figure 7: Direct refresh (trigger table).

definition (DDL) which is usually set at the start of the project and the interface code (PL/SQL Packages). New projects allow for closer integration of the change queue requirements with the interface code.

We recently installed a unified messaging system.[7] Although this system was supposed to use our existing Exchange server (the one discussed in *Embracing and Extending Windows 2000* [4]), our initial deployment required a second Exchange server, and in fact, its own Windows 2000 domain. An obvious step was to set up another listener in parallel to the one used for our primary Windows 2000 domain. However, along with the LOGINS information needed, we also needed voice mail specific information.

Since this was a new project, we were able to design the system so that all access to the "voice mail" tables was via single interface package. This allowed us to call the Meta_Change_Rtn.Log_XXX routines directly as needed. This gave us much greater flexibility in what we send to the Unity system for processing. For example, we have two "owners" for many objects. We have the "Unity Owner" which controls some access on the Unity system itself, and "System owner," which controls administrative access on the central database. For operational reasons, these often different entities. A voice mail tree will be administratively owned by a department, while on the Unity system, it will be "owned" by a group of administrators. We often need to change the administrative owner, but there is no need to send any changes to Unity. By having the single interface, this can be handled properly in the interface package.

In order to manage what tables are available to the listeners, we defined another database table, Meta_Change_Tables (Table 3) to hold that information. The primary purpose of this table is to document what is available. Most of this information is set when the table is defined, but one aspect is collected automatically. The first time a change record is logged for a specific Tname SubType pair, the PL/SQL call stack is saved to this table. This is a traceback of what procedures and packages called the logging routine. This can be very handy when tracing odd entries. This value will get refreshed if the Stack_Date value is cleared. This table also provides a handy selection list of possible tables when setting up listeners.

Defining Listeners

It generally doesn't do any good to talk, if no one is listening. There are three parts to each listener, an entry in the Meta_Change_Listeners table (Table 4), a listener specific interface package (such as the Insite_Interface package mentioned previously) and the actually interface application, be it an import table or a custom application. Like the Meta_Change_Tables, we also record the call stack of whoever calls for this listener. Although we have some concept of role based access control built in for each queue, in all of our deployments so far, we have written a specific interface package which provides the access control we need.

[7]Cisco Unity – voice messages and email are co-mingled on an Exchange server, with access to both via both the telephone and Outlook or other email agents

Table	sub type	Description	Count
BUILDINGS		Buildings (from INSITE Space Management)	258
DEPARTMENT		Departments from the phone directory	3442
GROUPS		Unix Groups	12
GROUP_MEM		Members of Unix Groups	871
INSITE_FLOOR		Floors within buildings (from Insite)	46
INSITE_SITE		Campuses (from Insite)	1
LOCATIONS		Rooms within Buildings (from Insite)	11890
LOGINS		Computer accounts (email)	61632
PERSON	Address	Address information	209016
PERSON	Dir_Orgn	Departmenal affiliation from directory	406
PERSON	Merge	Database cleanup – really ugly	224
PERSON	PEOPLE	Basic person information, Name, DOB, ID Numbers	160850
PERSON	Status	Current status for a person (Student, Employee, etc.)	95468
PERSON	Telephone	Telephone number (home, campus, etc.)	78960
PERSON	UDI	*User Directory Information*: Class Year, web page, email address	5725
UNITY_VMAIL		Command for Unity Voice Messaging System.	6072

Table 2: Tables and sub type.

```
Create or Replace Trigger Directory_Telephone_Trig_Upd
 after update on Directory_Telephone for each row
begin
   Meta_Change_Rtn.Log_Update( tname => 'PERSON',
        subtype => 'Telephone', rrowid => :new.rowid,
        person_id => :new.Person_Id, Aux_String => :new.tele_type);
end  Directory_Telephone_Trig_Upd;
```

Figure 8: Telephone change trigger.

We currently have seven listeners defined (Table 5). Of those, three are "real time," polling for changes frequently, and the others get once a day updates. In addition, both BEST and CMMS have interactive tools available to push through individual records on demand.

Linking Listeners with Tables

The last part of the puzzle is the Meta_Change_Interests table (Table 6) which defines which table and subtype pairs any given listener is interested in. This mapping is maintained with a web based tool, making it very easy to maintain these relationships. This tool also allows you to display pending and processed change counts, flush pending records (handy during development), as well as the call stacks for tables and listeners.

When a call is made to one of the Meta_Change_Rtn.Log_XXX routines, it takes the Tname and Subtype parameters, looks for listeners in the Meta_Change_Interests table (Table 6) that are interested,

Field	Type	Description
TNAME	varchar2(32)	Primary Key(1) – The table we are monitoring. Matches Meta_Change_Queue.Tname.
SUBTYPE	varchar2(32)	Primary Key(2) – An optional subtype of the table.
COMMENTS	varchar2(255)	A short description of what we are logging. Intended to help developers.
CALL_STACK	varchar2(2000)	The formatted "call stack" that made the a log entry. This is set when Stack_Date is null.
STACK_DATE	date	The date when the latest call stack was recorded. This will trigger refresh of the call stack data.
PERSON_ID	varchar2(65)	The source (if any) of the person_id value.
PKEY_STRING	varchar2(255)	The source of the pkey_string. This may be a composite value.
PKEY_NUMBER	varchar2(255)	The source, if any for the pkey_number. These values are generally not person_id values.
AUX_STRING	varchar2(255)	The source of the aux_string. This may be a composite value.

Table 3: Meta_Change_Tables definition.

Field	Type	Description
PROC_NAME	varchar2(8)	Primary Key(1) – The name of the valid listener. Used in the Get_Changes function call.
COMMENTS	varchar2(1024)	A description of what this listener is.
ROLE	varchar2(32)	An optional Oracle role needed to access this queue.
OWNER	number	The simon.people.id of who "owns" this queue.
CALL_STACK	varchar2(2000)	The formatted "call stack" that made the a log entry. This is set when Stack_Date is null.
STACK_DATE	date	The date when the latest call stack was recorded. This will trigger refresh of the call stack data.

Table 4: Meta_Change_Listeners definition.

Listener	Count	Frequency	Description
ADSI	25566	5 Sec	Active Directory – our primary windows 2000 domain.
Applix	148401	Daily	The trouble ticketing system for the computer center.
BEST	52633	Daily	ID card and physical access control system
CMMS	162520	Daily	Physical plant trouble ticket and payroll system.
Insite	52407	Daily	Space Management system (OFMS)
LDAP	176774	5 Sec	Directory service
Unity	16573	3 Min	Unified voice and email messaging system

Table 5: Current listeners.

Field	Type	Description
PROC_NAME	varchar2(8)	The name of the listener.
TNAME	varchar2(32)	The table that the listener (Proc_Name) is interested in.
Subtype	Varchar(32)	The subtype if applicable.
COMMENTS	varchar2(1024)	Maybe a reason WHY it is interested.

Table 6: Meta_Change_Interests definition.

and for each one, makes an entry in the Meta_Change_Queue table (Table 7). The name of the listener is set in both the Queue_Name and Proc_Name fields. When a record is processed, the Queue_Name column will be set to null. By putting an index on this field, and clearing it when it has been processed, the calls to Get_Changes can be done very quickly and efficiently.

Conclusions

At present, we have seven distinct "listeners" waiting for changes in one or more of 16 defined tables and sub types. To date, this system has processed over a half million changes. The three "real time" polling processes do not appear to put any noticeable load on the database, and in fact we have several other similar polling processors handling password changes, and they also do not noticeably load our database server. The approach of using an index on a key column that is cleared when the record has been processed works very effectively, and we will continue to use that here and with other processes, such as our password synchronization for our "single signon." We recently modified our password processing (described in [4]) to re-encrypt a password change for additional authentication realms (LDAP, Kerberos version 5, and our second Active Directory domain for the Unity Voice mail system.)

The import table/trigger approach has been very handy in providing interactive response to some of our processes and will likely be our interface method of choice when dealing with new Oracle based vendor applications, as well as internally developed applications where we want to maintain that clear demarcation between systems.

Futures

This Meta Change Queue system is fully operational and well integrated with our environment. I don't currently plan any major changes to it, but we will be making minor changes as new systems come along.

XML Output

Currently, each new listener required a listener specific interface package to be written. One area that may be worth exploring is a generic listener that generates XML. This will most likely happen when we

Field	Type	Description
TNAME	varchar2(32)	The name of the table (real or conceptual) that has been changed.
SUBTYPE	varchar2(32)	The subtype of this table – if any.
RROWID	rowid	The rowid of the record that was changed. This may be useful in speeding processing.
CHANGE_TYPE	varchar2(1)	The type of change; "I" – Insert, "C" – Change, "D" – Deletion. Some indication of what happened to the record.
PERSON_ID	number	The Simon.People.Id (if any). This is often a primary key for Simon tables.
PKEY_STRING	varchar2(32)	A varchar2 primary key value, for tables that do not use Person_Id as their key. This is optional.
PKEY_NUMBER	number	A numeric primary key value, similar to Pkey_String, only numeric rather than varchar2.
AUX_STRING	varchar2(255)	An optional extra value that might be useful the receiving system. This might be the old name.
ENTRY_DATE	date	The sysdate value when this change entry was made.
ENTRY_NUMBER	number	An ever increasing sequence number. This can be used to order changes.
PROC_DATE	date	The date when this record was processed and could be cleared.
HOLD_UNTIL	date	The time and date when this record should again be made available for processing. This can be used by other systems that can't process an event now, but want to get it eventually. Some other process will need to requeue these entries.
QUEUE_NAME	varchar2(8)	The name of the listener who is waiting for this record. This is the trigger value for pending entries. This column is indexed, and once a record is processed, this should be set to null. This will keep the index small and fast, allowing for low overhead and frequent polls.
PROC_NAME	varchar2(8)	The name of the listener. Initially, it is the same as the Queue_Name, but Queue_Name will be cleared after processing, this helps us track which listener got this record.
RETRY_COUNT	number	The number of times that this record was "put back" by the listener. This can help identify problem records and allow for back off options using the hold_until feature.

Table 7: Meta_Change_Queue definition.

get a new system that can accept an update stream in a format like that. Given the existing examples and support code, development of these interface packages has not been a problem. They are generally pretty simple and straightforward.

Status Reporting

We now have listeners automatically tied into to our process monitoring system, which will report on overall system problems. However, we have not done much with record level feedback and error reporting. In general, problem records don't get processed and cycle around for a while until someone notices them and takes appropriate action. This hasn't been much of a problem, but is something we need to look at more closely.

One of the objectives of my division, is to provide metrics for our activities. We are currently logging some periodic summaries of changes, and more formal analysis and reporting would be desirable.

Other Listeners

New listeners are generally prompted by the arrival of new systems, and as these systems are generally from other divisions, is not easy to predict what and when. We do have some existing systems that could benefit from the Meta Change Queue approach, and we will be exploring these areas. Some of them include:

- DNS configuration – providing end user tools for DNS changes, with immediate changes going via the MCQ.
- DHCP configuration – this has proven to be a "growth area" as we need to implement ways of rapidly change DHCP configuration as the result of virus scans, abuse investigations and so on.

Bulk Priority Queue

I will be adding a low priority queue, that will allow bulk entries to flow when "real time" requests are not pending. This has become an issue when mass create jobs "lock up" a listener for a long time and interactive users are trying to work. This change will be done entirely within the Get_Changes routine and none of the listeners will need to be changed.

References and Availability

Some of the examples in this paper have been edited for publication, frequently, some of the error handling code has been removed. While this should not impact your understanding of how this works, if you are going to implement something like this, I would suggest looking at the actual source code to see some of the special cases that we had to deal with. Some are very site specific, but will give you some idea of some of the details we had to handle.

This project is part of (but not dependent on) the Simon system, an Oracle based system used to assist in the management of our computer accounts [4], enterprise white pages [3], printing configuration [2], All source code for the Simon system, is available on the web. See http://www.rpi.edu/campus/rpi/simon/README.simon for details. In addition, all of the Oracle table definitions as well as PL/SQL package source are available at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html .

Acknowledgements

I would like to thank Andy Mondore for reviewing this paper. Special thanks also go to Alan Powell, Mike Douglass, Rich Bogart and Chet Burzynski all of RPI and also Lance Holloway of BEST Access Systems and Megan Whyman of OFMS for their contributions to this project. I also want to thank Rob Kolstad for his excellent (as usual) job of typesetting this paper.

Author Biography

Jon Finke graduated from Rensselaer in 1983 with a BS-ECSE. After stints doing communications programming for PCs and later general networking development on the mainframe, he then inherited the Simon project, which has been his primary focus for the past 13 years. He is currently a Senior Systems Programmer in the Networking and Telecommunications department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems. More recently, Jon has taken on support of the Telecommunications billing system,[8] and providing data and interfaces for Unity Voice Messaging and some Voice over IP projects. When not playing with computers, you can often find him merging a pair of adjacent row houses into one, or inventing new methods of double entry accounting as treasurer for Habitat for Humanity of Rensselaer County. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

References

[1] Armstrong, Eric, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt, *Oracle 7 Server Application Developer's Guide*, Chapter 8, Oracle Corporation, pp. 1-29, December, 1992.

[2] Finke, Jon, "Automating Printing Configuration," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, USENIX, pp. 175-184, September, 1994.

[3] Finke, Jon, "Institute White Pages as a System Administration Problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp. 233-240, USENIX, October, 1996.

[4] Finke, Jon, "Embracing and Extending Windows 2000," *The Sixteenth Systems Administration Conference (LISA 2002)*, USENIX, November, 2002.

[5] Finke, Jon, "Process Monitor: Detecting Events That Didn't Happen," *The Sixteenth Systems Administration Conference (LISA 2002)*, pp. 145-153, USENIX, November, 2002.

[8] AXIS – Pinnacle CMS by Paetec

Solaris Zones: Operating System Support for Consolidating Commercial Workloads

Daniel Price and Andrew Tucker – Sun Microsystems, Inc.

ABSTRACT

Server consolidation, which allows multiple workloads to run on the same system, has become increasingly important as a way to improve the utilization of computing resources and reduce costs. Consolidation is common in mainframe environments, where technology to support running multiple workloads and even multiple operating systems on the same hardware has been evolving since the late 1960's. This technology is now becoming an important differentiator in the UNIX and Linux server market as well, both at the low end (virtual web hosting) and high end (traditional data center server consolidation).

This paper introduces Solaris Zones (*zones*), a fully realized solution for server consolidation projects in a commercial UNIX operating system. By creating virtualized application execution environments within a single instance of the operating system, the facility strikes a unique balance between competing requirements. On the one hand, a system with multiple workloads needs to run those workloads in isolation, to ensure that applications can neither observe data from other applications nor affect their operation. It must also prevent applications from over-consuming system resources. On the other hand, the system as a whole has to be flexible, manageable, and observable, in order to reduce administrative costs and increase efficiency. By focusing on the support of multiple application environments rather than multiple operating system instances, zones meets isolation requirements without sacrificing manageability.

Introduction

Within many IT organizations, driving up system utilization (and saving money in the process) has become a priority. In the lean economic times following the post dot-com downturn, many IT managers are electing to adopt server consolidation as a way of life. They are trying to improve on typical data center server utilizations of 15-30% [1] while migrating to increasingly commoditized hardware. But the cost savings promised are not always realized [12]. Consolidation can drive down initial equipment cost, but it can also increase complexity and recurring costs in several ways. In our experience, this has made many system administrators reluctant to embrace consolidation projects. We believe that when implemented effectively, consolidation can free system administrators and IT architects to pursue higher service levels, better overall performance, and other long term projects. With an appropriate solution, greater specialization (and in turn, higher expertise) can be achieved; some administrators can focus on the maintenance of the physical platforms, and others can concentrate on the deployment of applications.

Administrators currently lack an all-in-one solution for server consolidation, as existing solutions require administrators to purchase, author, or deploy additional software. This paper explains how a server consolidation facility tightly integrated with the core operating system can provide answers to these problems.

Barriers to Consolidation

Consolidation projects face a variety of technical problems. First and foremost, applications can be mutually incompatible when run on the same server. In one real-world example, two poorly written applications at a customer site both wanted to bind a network socket to port 80. While neither application was a substantial resource user, the customer resolved the conflict by buying two servers! Applications can also be uncooperative when administrators wish to run multiple instances of the same application on the same node. For example, dependencies on running as a particular user ID can make it difficult to distinguish one running instance from another. Hard-coded log file locations or other pathnames can make it difficult to deploy two distinct versions of a particular application on the same node. At the highest level, solving this problem requires some form of *namespace isolation*, allowing administrators to make applications unaware of the presence of others. In the customer's example, deploying to two separate OS instances running on two separate systems provides complete namespace isolation, but the cost is very high.

A second technical problem faced by consolidators is security isolation. If multiple applications are to be deployed on a single host, what if there is a security bug in one of the applications? Even if each application is running under a different user ID (except for the applications that demand to run as `root`!), a wily intruder may be able to embark on a *privilege escalation*, in which the successively achieves higher levels of privilege until the entire system is compromised. If administrators are unable to assess the extent of the damage, the consolidated system might require a rebuild. Ideally, one should be able to create namespaces that are at

fundamentally reduced levels of privilege: root in such an environment should be less powerful than the traditional UNIX root.

If a particular workload is compromised, a protective mechanism that wards off denial of service and resource exhaustion attacks against the rest of the system should be in place. Similarly, consolidation projects must address quality of service guarantees and must often be able to account for resource utilization for billing or capacity planning purposes. Existing resource management solutions address many of these requirements by providing resource partitioning, advanced schedulers, and an assortment of resource caps, reservations, and controls. However, these facilities do not typically offer security and namespace isolation and in cases where both are available, they have not been closely integrated.

A Comprehensive Solution

While a range of solutions exists to each of the problems described above, we discovered that no comprehensive consolidation facility was available as a core component of a commonly available operating system. We determined that deeper integration and a more "baked in" facility for consolidation would allow administrators to approach consolidation projects without the burden of designing the infrastructure to do so from component pieces. As a design goal, we established that administrators should need only a few minutes and a very few configuration choices to instantiate and start up a new application container, which we dubbed a *zone*. We also wanted our project to be a pure software solution that would work on a variety of hardware platforms, with the least possible performance tax.

At the highest level, zones are lightweight "sandboxes" within an operating system instance, in which one or more applications may be installed and run without affecting or interacting with the rest of the system. They are available on every platform on which Solaris 10 runs: AMD64, SPARC64, UltraSPARC, and x86. Applications can be run within zones with no changes, and with no significant performance impact for either the performance of the application or the base operating system.

Outline

This paper introduces zones and explains how we built a server consolidation facility directly into a production operating system, Solaris 10. The next sections describe related work, an overview of the facility, our design principles, and the architectural components of the project. The paper then explores specific aspects of the zones implementation, including resource management, observability and performance. We also discuss experiences to date with the facility.

Related Work

Much of the previous work on support for server consolidation has involved running multiple operating system instances on a single system. This can be done either by partitioning the physical hardware components into disjoint, isolated subsets of the overall system [5, 8], or by using virtual machine technologies to create abstracted versions of the underlying hardware [2, 7, 14]. Hardware partitioning, while providing a very high degree of application isolation, is costly to implement and is generally limited to high-end systems. In addition, the granularity of resource allocation is often poor, particularly in the area of CPU assignment. Virtual machine implementations can be much more granular in how resources are allocated (even time-sharing multiple virtual machines on a single CPU), but suffer significant performance overheads. With either of these approaches, the cost of administering multiple operating system instances can be substantial.

More recently, a number of projects have explored the idea of virtualizing the operating system's application execution environment, rather than the physical hardware. Examples include the Jails facility in FreeBSD [9] and the VServer project available for Linux systems [13]. These efforts differ from virtual machine implementations in that there is only one underlying operating system kernel, which is enhanced to provide increased isolation between groups of processes. The result is the ability to run multiple applications in isolation from each other within a single operating system instance. This should result in reduced administration costs, since there is only one operating system instance to administer (patch, backup, etc.); in addition, the performance overhead is generally minimal. Such technologies can also be used to create novel system architectures, such as the distributed network testbed provided by the PlanetLab project [3].

These technologies can be used as "toolkits" to assemble point solutions to virtualization problems, but at present they lack the comprehensive support required for supporting commercial workloads. The barrier to entry for administrators is also high due to the lack of tools and integration with the rest of the operating system.

Zones Overview

Zones provides a solution which virtualizes the operating system's application environment, and leverages the performance and sharing possible. At the same time, we have provided deeper and more complete system integration than is typical of such projects. We have been gratified when casual users mistake the technology for a virtual machine. This section provides a broad overview of the zones architecture and operation.

Figure 1 provides a block diagram of a system with four zones, representing a hypothetical consolidation. Zones *red*, *neutral* and *lisa* are *non-global zones*

running disjoint workloads. This example demonstrates that different versions of the same application may be run without negative consequences in different zones to match the consolidation requirements. Each zone can provide a rich (and different) set of customized services, and to the outside world, it appears that four distinct systems are available. Each zone has a distinct root password and its own administrator.

Basic process isolation is also demonstrated; a process in one non-global zone cannot locate, examine, or signal a process in another zone. Each zone is given access to at least one logical network interface; applications running in distinct zones cannot observe the network traffic of the other zones even though their respective streams of packets travel through the same physical interface. Finally, each zone is provided a disjoint portion of the file system hierarchy, to which it is confined.

The *global* zone encloses the three non-global zones and has visibility into and control over them. Practically speaking, the global zone is not different from a traditional UNIX system; root generally remains omnipotent and omniscient. The global zone always exists, and acts as the "default" zone in which all processes are run if no non-global zones have been created by the administrator.

We use the term *global administrator* to denote a user with administrative privileges in the global zone. This user is assumed to have complete control of the physical hardware comprising the system and the operating system instance. The term *zone administrator* is used to denote a user with administrative privileges who is confined to the sandbox provided by a particular non-global zone.

Managing zones is not complicated. Figure 2 shows how to create a simple, non-networked zone called *lisa* with a file system hierarchy rooted at /aux0/lisa, install the zone, and boot it. Booting a zone causes the init daemon for the zone to be launched. At that point, the standard system services such as cron, sendmail, and inetd are launched.

Design Principles

This section and the next examine the zones architecture in greater depth; before doing so it helps to examine the design principles we applied. First and foremost, our solution must *solve consolidation problems* such as those highlighted in the first section. The solution must provide namespace isolation and abstraction, security isolation, and resource allocation and management.

Second, the facility must *support commercial applications*: these are often scalable, threaded, highly connected to the network via TCP/IP, NFS, LDAP, etc. These applications come with installers and usually interoperate with the packaging subsystem on the host. More importantly, because these applications are often

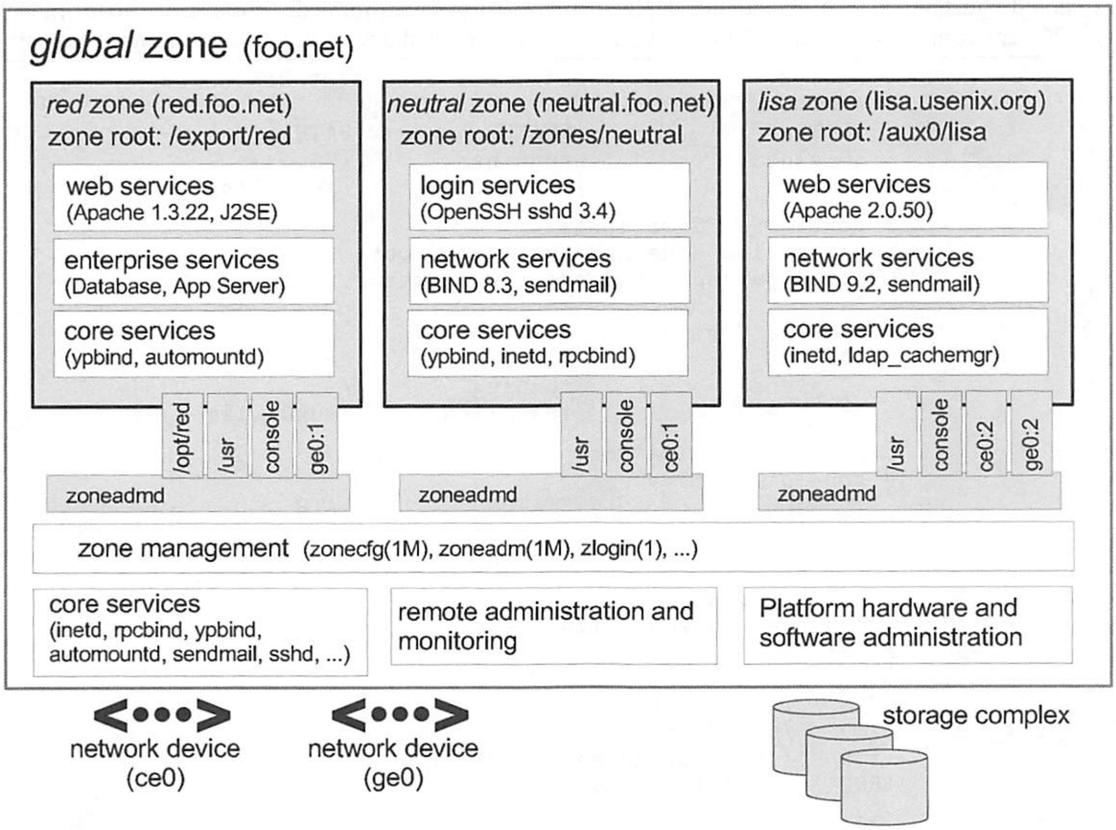


Figure 1: Zones block diagram.

opaque in operation, they should work "out of the box" within a zone whenever possible. Software developers should not need to modify applications, and administrators should not need to develop scripting wrappers or have a deep understanding of UNIX internals to deploy these applications. Similarly, administrators interacting with this facility should be *pilots, not mechanics*. As much as possible, system administrators should be able to view the application environment as a vehicle for deploying applications, not as a collection of parts to assemble. Setup should be simple and the entire system should look and feel as much like a normal host as possible. In addition, the solution should *enable delegation* wherever possible. The administrator of the global zone should be able to configure the overall system and delegate further control to zone administrators.

By *exploiting sharing and semantics* inside a single operating system instance, we can support a large number of application environments with relatively few resources. Operating in a shared environment means that monitoring application environments can be performed transparently. For example, from the pilot's seat, we should immediately be able to tell which process on the system (regardless of the application environment in which it runs) is using the most CPU cycles.

The solution must *scale and perform* with with the underlying platform. A 64-CPU application environment should "just work," as should the deployment of 20 environments on a 1-CPU system. Additionally, the solution should levy little or no performance tax on applications run inside it. Finally, minimal performance impact should be present on a system with no application environments.

To address these design principles, we divided the zones architecture into five principal components.

- A state model that describes the lifecycle of the zone, and the actions that comprise the transitions.
- A configuration engine, used by administrators to describe the future zone to the system. This allows the administrator to describe the "platform," or those parameters of the zone that are controlled by the global administrator, in a persistent fashion.
- Installation support, which allows the files that make up the zone installation to be deployed into the *zone path*. This subsystem also enables patch deployment and upgrades from one operating system release to another.
- The *application environment*, the "sandbox" in which processes run. For example, in Figure 3 each zone's application environment is represented by the large shaded box.
- The virtual platform, comprised of the set of platform resources dedicated to the zone.

We'll explore these subsystems in more depth in subsequent sections.

Zones State Model

A well-formed, observable state model that describes the zone lifecycle is an important part of the

```
# zonecfg -z lisa 'create; set zonepath=/aux0/lisa'
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   - lisa             configured     /aux0/lisa

# zoneadm -z lisa install
Constructing zone at /aux0/lisa/root
Copying packages and creating contents file
...
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   - lisa             installed      /aux0/lisa

# zoneadm -z lisa boot
# zoneadm list -vc
  ID NAME             STATUS         PATH
   0 global           running        /
   7 lisa             running        /aux0/lisa

# zlogin lisa
[Connected to zone 'lisa' pts/7]
zone: lisa
# ptree
1716   /sbin/init
1769   /usr/sbin/cron
1775   /usr/lib/sendmail -Ac -q15m
1802   /usr/lib/ssh/sshd
...
```

Figure 2: Zones administration.

pilot model design principle; it makes it easier for administrators to manage the zones present on the system. Figure 3 illustrates the zone state model. While this is of interest to the global administrator, zone administrators need not be aware of these states. A zone can be in one of four primary states, or in one of several secondary, or transitional, states:

CONFIGURED: A zone's configuration has been completely specified and committed to stable storage.

INSTALLED: Based on the zone's configuration, a unique root file system for the zone has been instantiated on the system.

READY: At this stage, the virtual platform for the zone has been established: the kernel has created the zsched process, network interfaces have been plumbed, file systems mounted, and devices configured. At this point, there are no user processes associated with the zone (zsched is a system process, and lacks a user address space).

RUNNING: The init daemon has been created and appears to be running. init will in turn start the rest of the processes that comprise the application environment.

SHUTTING DOWN: The zone is transitioned into this state when either the global or non-global zone administrator elects to reboot or halt the zone. The zone remains in this state until all user processes associated with the zone have been destroyed.

DOWN: The zone remains in this state until the virtual platform has been completely destroyed: filesystems and NFS shares are unmounted, IPC objects destroyed, network interfaces unplumbed, etc. At that point the zone returns to the INSTALLED state.

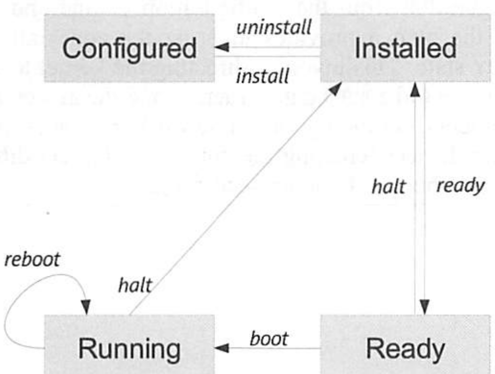


Figure 3: Zones state model.

Configuration Engine

Zones present a simple and concise configuration experience for system administrators. A command shell, zonecfg, is used by the global administrator to configure the zone's properties and describe the zone to the system. The tool can be used in interactive mode or scripted to create a new zone or edit existing zones. The configuration includes information about the location of the zone in the file system, IP addresses, file systems, devices, and resource limits. The zone configuration is retained by the system in a private repository (presently, an XML file), and keyed by zone name.

The design of zonecfg was challenging: Zone configurations can be complex, but we wanted to instantiate a new zone with a minimum number of commands and without having to navigate through a complex configuration file. Ultimately, the only mandatory parameter is the zonepath – the location in the file system where the zone should be created.

Installation Support

The zones installer is an extension to the Solaris install and packaging tools. An important goal was to be able to create a zone on an existing system, without needing to consult installation media. Binary files such as /usr/bin/ls can simply be copied from the global zone, or imported to the zone using a loopback mount to save disk space.

Files which are customizable by an administrator, such as /etc/passwd, must not be copied from the global zone into the zone being installed. Such files must be restored to their "factory default" condition. In order to accomplish this, the installer archives private, pristine copies of such volatile and editable system files when the global zone itself is installed or upgraded. The zones installer uses these archived versions when populating zones.

Because the zone installer is package-aware, the end result of zone installation is a virtual environment with an appropriately populated package database. This means that packaging utilities such as pkgadd can be used by the zone administrator to add or patch unbundled or third-party software inside the zone while also allowing the global administrator to the correctly upgrade and patch the system as a whole.

Application Environment

The application environment forms the core of the zones implementation. Using the facilities it provides, other subsystems such as NFS, TCP/IP, file systems, etc. have been "virtualized," that is, rearchitected to be compatible with the zones design.

At the most basic level, the kernel identifies specific zones in the same fashion as it does processes, by using a numeric ID. The *zone ID* is reflected in the cred and proc structures associated with each process. The kernel can thus easily and cheaply determine the zone membership of a particular process. This mapping is at the heart of the implementation. We have also found that virtualizing kernel subsystems (for example, process accounting) is often not terribly difficult if the subsystem's global variables are lifted up into a per-zone data structure (in other cases, such as TCP/IP, the virtualization required is more pervasive).

The process of booting the application environment is similar to the late stages of booting the operating system itself. In the kernel, a special process,

zsched, is created. This mimics the traditional UNIX process 0, sched. When seen from inside a zone, zsched is at the root of the process tree. zsched also acts as a container for a variety of per-zone data that is hard to express in other ways. RPC thread pools and other per-zone kernel threads, as well as resource controls and resource pool bindings, are handled in this fashion. Next, the init daemon is formed, associated with the zone, and exec'd to set it running in userspace; init then initiates the process of starting up other services that make the zone behave like a standalone computer system.

Zones are also assigned unique identities. The zone name, which is used to label and identify the zone, is assigned by the global administrator. Control of the node name, RPC domain name, Kerberos configuration, locale, time zone, root password and name service configuration is entirely delegated to the zone administrator. When a zone is first booted, the zone administrator is stepped through the process of setting up this configuration via an interactive tool.

Security concerns are central to the design of the application environment. Fundamentally, a zone is less powerful than the global environment, because zones take advantage of the fine-grained privilege mechanism available in Solaris 10 [11]. This mechanism changes the traditional all-or-nothing "super-user" privilege model into one with distinct privileges that can be individually assigned to processes or users.[1] A zone runs with a reduced set of privileges, and this helps to ensure that even if a process could find a way to escape namespace isolation enforced by the zone, it would still be constrained from escalating to higher privilege. For example, writing to /dev/kmem requires all privileges. All non-global zone processes and their descendants have fewer than all privileges, and are constrained from ever achieving all privileges, so the kernel will never allow such a process to write to /dev/kmem. The namespace isolation facilities provided by zones coupled with privilege containment provide a sound double-hulled architecture for secure operation. Although the set of privileges available in a zone is currently fixed, we plan to make this configurable in the future; this will allow administrators to create special-purpose zones with only the minimal set of privileges needed to run a particular service.

One significant design challenge the project faced was: how can we cross the boundary between global and non-global zones in a safe fashion? We authored the zlogin utility to allow global administrators to descend into specific zones; this command is modeled after familiar utilities such as rlogin. The process of transferring a running process from one zone to another is complex, and was a challenging aspect of the implementation. We took care to prevent any data from "leaking" from the global zone into non-global zones; this required sanitization of parent process IDs, process group IDs, session IDs, credentials, fine-grained privileges, core file settings, and other process model-related attributes. Processes whose parent process lies outside the zone (as is the case with zlogin to a zone) are faked within the zone to have zsched's PID as their parent process ID. Similarly, signals sent from the global zone to non-global zone processes appear to originate from zsched.

Virtual Platform

The virtual platform is the "bottom half" of a zone. Conceptually, it is comprised of the physical resources that have been made available to the zone. The virtual platform is also responsible for boot, reboot and halt, and is managed by the zoneadmd daemon.

The virtual platform takes a snapshot of the zone configuration from the configuration engine and follows the plan it provides to bring the zone into the READY state. This involves directing the kernel to create the central zone_t data structure and the zsched kernel process, setting up virtual network interfaces, populating devices, creating the zone console, and directing any other pre-boot housekeeping.

[1] This is similar to the capability feature available in Linux.

```
# zlogin -C lisa
[Connected to zone 'lisa' console]

lisa console login: root
Password:

# reboot
Aug 13 14:44:07 lisa reboot: rebooted by root

[NOTICE: Zone rebooting]

SunOS Release 5.10 Version s10_65 64-bit
Copyright 1983-2004 Sun Microsystems, Inc. All rights reserved.
Use is subject to license terms.
Hostname: lisa
NIS domain name is usenix.org

lisa console login: ~.
[Connection to zone 'lisa' console closed]
#
```

Figure 4: Zones console.

Uniquely, the zone console can exist even before the zone is in the ready state. This mimics a serial console to a physical host, which can be connected even when a machine is halted, and it provides a familiar experience for administrators. The console itself is a STREAMS driver instance that is dynamically instantiated as needed. It shuttles console I/O back and forth from the zone (via /dev/console) to the global zone (via /dev/zcons/<zonename>/masterconsole). zoneadmd then acts as a console server to the zlogin -C command. Figure 4 shows a typical console session. We found that the zone pseudo-console was a key to helping users see that a zone is a substantially complete environment, and perhaps more importantly, a *familiar* environment.

Virtualization of Specific Subsystems

One of the principal challenges of the zones project was making decisions about the "virtualization strategy" for each kernel subsystem. Generally, we sought to allow the global administrator to observe and control the entire system. But this was not always possible due to API restrictions (for example, APIs dictated by a particular standard), implementation constraints, or other factors. The next sections detail the virtualization that was required for each primary kernel subsystem.

Process Model

One of the basic principles of zones is that processes in non-global zones should not be able to affect the activity of processes running within another zone. This also extends to visibility; processes within one (non-global) zone should not even be able to see processes outside that zone, and by extension should not be able to observe the activity of such processes. This is enforced by restricting the process ID space exposed through the /proc file system and process-specific system calls such as kill, priocntl, and signal. If the calling process is running within a non-global zone, it will only be able to see or affect processes running within the same zone; applying the operations to process IDs in any other zone will return an error. The error code is the same as the one returned when the specified process does not exist, to avoid revealing the fact that the selected process ID exists in another zone. This policy also ensures that an application running in a zone sees a consistent view of system objects; there aren't any objects that are visible through some means (e.g., when probing the process ID space using kill) but not others (e.g., /proc).

The dual role of the global zone, acting as both the default zone for the system and as the nexus of system-wide administrative control, raises some interesting issues. Since applications within the zone have access to processes and other system objects in other zones, the effect of administrative actions may be wider than expected. For example, service shutdown scripts often use pkill to signal processes of a given name to exit. When run from the global zone, all such processes in the system, regardless of zone, will be signaled.

On the other hand, the system-wide scope is often desired. For example, an administrator monitoring system-wide resource usage would want to look at process statistics for the whole system. A view of just global zone activity would miss relevant information from other zones in the system that may be sharing some or all of the system's resources. Such a view is particularly important when the use of relevant system resources such as CPU, memory, swap, and I/O is not strictly partitioned between zones using resource management facilities.

We chose to allow any processes in the global zone to observe processes and other objects in non-global zones. This allows such processes to have system-wide observability. The ability to control or send signals to processes in other zones, however, is restricted by a fine-grained privilege, PRIV_PROC_ZONE. By default, only the root user in the global zone is given this privilege. This ensures, for example, that user tucker, whose user ID in the global zone is 1234, cannot kill processes belonging to user dp, whose user ID in the lisa zone is also 1234. Because different zones on the same system can have completely different name service configurations, this is entirely possible. The root user can also drop this privilege, restricting activity in the global zone to affect only processes in that zone.

Accounting and Auditing

Process and workload accounting provide an excellent example of both the challenges and opportunities for retrofitting virtualization into an existing subsystem. Accounting outputs a record of each process to a file upon its termination. The record typically includes the process name, user ID, exit status, statistics about CPU usage, and other billing-related items. The UNIX System V accounting subsystem, which remains in wide usage, employs fixed size records that cannot be extended with new fields. Thus, we modified the system so that accounting records generated in any zone (including the global zone) only contain records pertinent to the zone in which the process executed. System V accounting can be enabled or disabled independently for each zone.

In addition, since Solaris 8, the system has provided a modernized "extended accounting" facility, with flexible record sizes. We modified this so that records are now tagged with the zone name in which the process executed, and are written *both* to that zone's accounting stream and to the global zone; this provides an important facility for consolidation, and, uniquely, the ability to account in detail for the activity of the application environment. The set of data collected, the location of the accounting record files, and other accounting controls may all be configured independently per-zone.

The Solaris security auditing facility has been similarly updated with the addition of a zonename

token. An audit record describes an event, such as writing to a file, and the stream of audit records is written to disk and may be processed later. Each zone can access the appropriate subset of the audit trail, and the global zone can see all audit records for all zones. Because the global zone can track audit events by zone name, a complete record of auditable events can be generated per-zone. We think this represents an exciting possibility for intrusion detection and analysis.

IPC Mechanisms

Local inter-process communication (IPC) represents a particular problem for zones, since processes in different (non-global) zones should normally only be able to communicate via network APIs, as would be the case with processes running on separate machines. It might be possible for a process in the global zone to construct a way for processes in other zones to communicate, but this should not be possible without the participation of the global administrator.

IPC mechanisms that use the file system as a rendezvous, such as pipes, STREAMS, UNIX domain sockets, doors, and POSIX IPC objects, fit naturally into the zone model without modification since processes in one zone will not have access to file system locations associated with other zones. Because the file system hierarchy is partitioned, there is no way for processes in a non-global zone to achieve rendezvous with another zone without the involvement of the global zone (which has access to the entire hierarchy).

The System V IPC interfaces allow applications to create persistent objects (shared memory segments, semaphores, and message queues) for communication and synchronization between processes on the same system. The objects are dynamically assigned numeric identifiers that can be associated with user-defined keys, allowing usage of a single object in unrelated processes. Objects are also associated with an owner (based on the effective user ID of the creating process unless explicitly changed) and permission flags that can be set to restrict access when desired. In order to prevent sharing (intentional or unintentional) between processes in different zones, a zone ID is associated with each object, based on the zone in which the creating process was running at time of creation. Non-global zone processes are only able to access or control objects associated with the same zone. An administrator in the global zone can still manage IPC objects throughout the system without having to enter each zone. The key namespace is also virtualized to be per-zone, which avoids the possibility of key collisions between zones.

Networking

As discussed earlier, each zone is configured with one or more IP addresses. For each address assigned, a logical network interface is created in the global zone when the zone is readied. This address is then assigned to the zone. The system as a whole looks like a traditional multi-home server, but internally the IP stack partitions the networking between zones in much the same way as it would be partitioned between separate servers. From the perspective of an external network observer, a system with booted zones appears to be set of separate servers.

Each IP address and its associated logical interface are dedicated for use by the assigned zone. Only processes within the zone can send packets from that address or receive packets sent to that address. Logical interfaces can share a physical network interface, however, so depending on how the zones are configured, different zones may wind up sharing network bandwidth on a single physical interface. The isolation of network traffic means that services such as sendmail, Apache, etc., can be run in different zones without worrying about IP port conflicts.

Applications in different zones on the same system can communicate using conventional networking, just as applications on different systems can communicate. This traffic is "short-circuited" within the IP stack rather than sending data over the wire, minimizing the communication overhead. One drawback is that existing firewalling products are not able to filter or otherwise act on cross-zone traffic, as it is handled entirely within IP and is not visible to any underlying firewalling products. We hope to remedy this in the future.

Sending and receiving broadcast and multicast packets is supported within any zone. Inter-zone broadcast and multicast is implemented by replicating outgoing and incoming packets as necessary, so that each zone that should receive a broadcast packet or each zone that has joined a particular multicast group receives the appropriate data.

Access to the network by non-global zones is restricted. The standard TCP and UDP transport interfaces are available, but some lower level interfaces, such as raw socket access (which allows the creation of IP packets with arbitrary contents) and DLPI are not. These restrictions are in place to ensure that a zone cannot gain uncontrolled access to the network, where it might be able to behave in undesirable ways. For example, a zone cannot masquerade as a different zone or host on the network. Access to ICMP is also supported, allowing popular utilities such as traceroute and ping to work properly.

The zones facility also provides support for manual configuration of IPv6 addresses, with support for automatic addressing planned for the future. Because much of the TCP/IP infrastructure is shared between all zones, some functionality is automatically supported and can be configured on behalf of a zone by the global administrator. For example, if IP Multipathing is configured within the global zone, the logical interfaces associated with a failed physical interface are automatically moved to a configured alternate interface. The individual zones do not need any

configuration to support this, and are not even aware of the failure.

IPsec and IPQoS facilities can be configured on behalf of a zone by the global administrator; in the future we hope to allow global administrators to delegate some of this configuration to non-global zones. It would also be convenient to provide DHCP client support so that the global zone could request IP addresses for non-global zones from a DHCP server, and work to incorporate this support is underway.

File Systems

We have seen that zones are rooted at a particular point in the file system. This is implemented in a fashion similar to the chroot system call, although that call's well known security limitations [6] are avoided and the zone is not escapable. Because a different mechanism is used, use of chroot is even possible within a zone.

When the zone boots, the configuration engine is consulted for a list of file systems to mount on behalf of the zone. These can include storage-backed file systems as well as pseudo-file systems. In particular, lofs, the Solaris loopback file system, provides a useful tool for constructing a file system namespace for a zone. It can be used to mount segments of a file system in multiple places within the namespace. For example, the /usr file system is typically loopback mounted read-only beneath the zone root. This results in a high degree of sharing of storage, and a freshly installed zone requires only about 60 MB of disk space. The use of loopback mounts also results in the sharing of process text pages in the buffer cache, further decreasing the impact of running large numbers of zones. However, this approach adds substantial complexity to the design and implementation of the packaging tools. For example, the zone installation software must be aware that a particular file system object such as /usr/bin/ls will be available, but it will not have to be copied to the zone's /usr file system.

Mounts require special handling as well. In Solaris /etc/mnttab is itself a mounted file system that, when mounted, exports the typical /etc/mnttab file. The mnttab handling code was modified so that each zone sees only the mounts accessible by it. As usual, the global zone can see everything.

A key security principle is that the global zone users should not be able to traverse the file system hierarchy of non-global zones. Allowing this would enable unprivileged users in the global zone to collaborate with root users in non-global zones. For example, a zone's root user might mark a binary in a zone setuid root, and collaborate with a non-root user in the global zone, who could then run the binary and gain superuser privileges. As such, the the zones infrastructure enforces that the zone root's parent directory be owned, readable, writable, and executable by root only. We were also careful to prevent zones components such as zlogin and zoneadmd from ever accessing files residing within zones, in order to avoid "traps" that might have been placed by privileged software within the zone.

Devices

A limited set of devices, accessible via the /dev hierarchy, are available to zones by default. Additionally, some devices required additional virtualization to support zones. The syslog device is a good example: each zone has a distinct /dev/log device with a separate message stream, so that syslog(3C) messages are delivered to the syslogd in the zone that generated them.

Administrators can use the configuration engine to add additional devices to the zone as needed. This carries additional security risk because device interfaces are relatively unconstrained. A single device driver can form its own subsystem of APIs and semantics. For example, writing to a disk, writing to /dev/null, and writing to /dev/kmem all have completely different effects and security implications. As a general principle, we discourage the placement of physical devices into zones, as there is wide opportunity for mischief. For example, disks or disk partitions can be assigned to a zone, but the preferred method is for the global administrator to assign only file systems, which provide more uniform, auditable semantics.

A driver bug or improperly guarded feature could allow a hacker to attack the kernel. As a result, all of the devices included in a zone by default were audited and tested for security problems. We also addressed more systemic security problems; for example, an *imported device node* may allow a hacker to attack the system. For this reason, zones lack the privilege to call mknod(2) to create device nodes. However, this problem is more pervasive. If a hacker caused an NFS server to export a device node that matched the major and minor number of /dev/kmem and caused the zone to mount this share, then the system could be compromised. To defend against this attack, all mounts initiated from within a zone are guarded by the nodevices mount option, which prevents the opening of device nodes present on the mount. Note that even without nodevices, such an attack would remain difficult, as the reduced privilege allotted to the zone does not allow writing to the kmem device under any circumstances.

A final category of attacks could be carried out against the software managing the /dev hierarchy that runs in the global zone as part of the virtual platform. In this case, both global and non-global zones require access to the /dev hierarchy. The solution is to build and manage /dev for the zone *outside* of the zone's file system hierarchy, and then use the lofs file system to loopback mount /dev into the zone. Additionally, the kernel prohibits the zone from making all but the most basic modifications to its /dev hierarchy. Permission, group, and ownership changes are permitted; other file system operations are not.

NFS

Virtualizing client-side NFS support presents a somewhat unique challenge. NFS is not only a file

system: It also has semantics that are dependent on the network identity (hostname, RPC domain, etc.) of the client. For example, an NFS share may be exported solely to a client with a specific host name. Since each zone has a separate network identity, NFS mounts in different zones on the same system must be handled separately. In particular, operations to file system mounts associated with a zone must have matching credentials. This allows lower-level code (such as the RPC transport code) to keep track of the zone associated with a specific operation, even if that operation is being performed asynchronously. As a consequence, NFS mounts in a non-global zone cannot be accessed from the global zone.

Another complication is the use of kernel threads. The Solaris NFS implementation maintains a pool of in-kernel threads to asynchronously read-ahead data before it is needed, which improves performance when large files are read sequentially. When multiple zones can be using NFS, the thread pools need to be maintained on a per-zone basis. This allows the number of threads in each pool to be managed independently (since different zones may have different requirements with respect to concurrency) and means that threads can be assigned credentials associated with the appropriate zone.

Resource Management

Most of the prior discussion has described the ways in which zones can be used to isolate applications in terms of configuration, namespace, security, and administration. Another important aspect of isolation is ensuring that each application receives an appropriate proportion of the system resources: CPU, memory, and swap space. Without such a capability, one application can either intentionally or unintentionally starve other applications of resources. In addition, there may be reasons to prioritize some applications over others, or adjust resources depending on dynamic conditions. For example, a financial company might wish to give a stock trading application high priority while the trading floor is open, even if it means taking resources away from an application analyzing overall market trends.

The zones facility is tightly integrated with existing resource management controls available in Solaris [10]. These controls come in three flavors: *entitlements*, which ensure a minimum level of service; *limits*, which bound resource consumption; and *partitions*, which allow physical resources to be exclusively dedicated to specific consumers. Each of these types of controls can be applied to zones. For example, a fair-share CPU scheduler can be configured to guarantee a certain share of CPU capacity for a zone. In addition, an administrator within a zone can configure CPU shares for individual applications running within that zone; these shares are used to determine how to carve up the portion of CPU allocated to the zone. Likewise, resource limits can be established on either a per-zone basis (limiting the consumption of the entire zone) or a more granular basis (individual applications or users within the zone). In each case, the global zone administrator is responsible for configuring per-zone resource controls and limits, while the administrator of a particular non-global zone can configure resource controls within that zone.

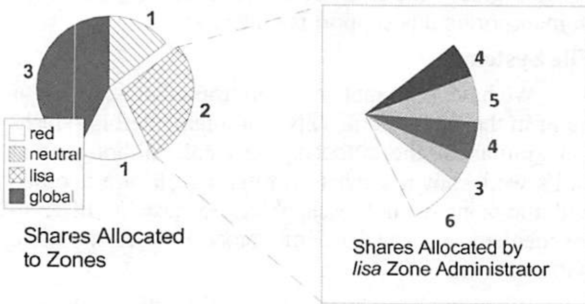


Figure 5: Zones and the fair-share scheduler.

Figure 5 shows how the fair-share CPU scheduler can be used to divide CPU resources between zones. In the figure, the system is divided into four zones, each of which is assigned a certain number of CPU shares. If all four zones contain processes that are actively using the CPU, then the CPU will be divided according to the shares; that is, the red zone will receive 1/7 of the CPU (since there are a total of seven shares outstanding), the neutral zone will receive 2/7, etc.. In addition, the lisa zone has been further subdivided into five *projects*, each of which represent a workload running within that zone. The 2/7 of the CPU assigned to the lisa zone (based on the per-zone shares) will be further subdivided among the projects within that zone according to the specified shares.

Resource partitioning is supported through a mechanism called *resource pools*, which allows an administrator to specify a collection of resources that will be exclusively used by some set of processes. Although the only resources initially supported are CPUs, this is planned to later encompass other system resources such as physical memory and swap space. A zone can be "bound" to a resource pool, which means that the zone will run only on the resources associated with the pool. Unlike the resource entitlements and limits described above, this allows applications in different zones to be completely isolated in terms of resource usage; the activity within one zone will have no effect on other zones. This isolation is furthered by restricting the resource visibility. Applications or users running within a zone bound to a pool will see only resources associated with that pool. For example, a command that lists the processors on the system will list only the ones belonging to the pool to which the zone is bound. Note that the mapping of zones to pools can be one-to-one, or many-to-one; in the latter case, multiple zones share the resources of the pool,

and features like the fair-share scheduler can be used to control the manner in which they are shared.

Figure 6 shows the use of the resource pool facility to partition CPUs among zones. Note that processes in the global zone can actually be bound to more than one pool; this is a special case, and allows the use of resource pools to partition workloads even without zones. Non-global zones, however, can be bound to only one pool (that is, all processes within a non-global zone must be bound to the same pool).

Performance and Observability

As noted in the related work section, one of the advantages of technologies like zones that virtualize the operating system environment over a traditional virtual machine implementation is the minimal performance overhead. In order to substantiate this, we have measured the performance of a variety of workloads when running in a non-global zone, when compared to the same workloads running without zones (or in the global zone). This data is shown in Figure 7 (in each case, higher numbers represent a faster run). The final column shows the percentage degradation (or improvement) of the zone run versus the run in the global zone. As can be seen, the impact of running an application in a zone is minimal. The 4% degradation in the time-sharing workload is primarily due to the overhead associated with accessing commands and libraries through the lofs file system.

Workload	Base	Zone	Diff (%)
Java	38.45	38.29	99.6
Time-sharing	23332.58	22406.51	96.0
Networking	283.30	284.24	100.3
Database	38767.62	37928.70	97.8

Figure 7: Performance impact of running in a zone.

We also measured the performance of running multiple applications on the system at the same time in different zones, partitioning CPUs either with resource pools or the fair share scheduler. In each case, the performance when using zones was equivalent, and in some cases better, than the performance when running each application on separate systems.

Since all zones on a system are part of the same operating system instance, processes in different zones can actually share virtual memory pages. This is particularly true for text pages, which are rarely modified. For example, although each zone has its own init process, each of those processes can share a single copy of the text for the executable, libraries, etc.. This can result in substantial memory savings for commonly used executables and libraries such as libc. Similarly, other parts of the operating system infrastructure, such as the directory name lookup cache (or DNLC), can be shared between zones in order to minimize overheads.

Observability Tools and Debugging

Because of the transparent nature of zones, all of the traditional Solaris /proc tools may be applied to processes running inside of zones, both from inside the non-global zone, and from the global zone. Additionally, numerous utilities such as ps, priocntl, ipcs, and prstat (shown in Figure 9) have been enhanced for zone-awareness.

In addition, we were able to enhance the DTrace [4] facility to provide zone context. In the following example, we can easily discover which zone is causing the most page faults to occur; see Figure 11.

We were pleasantly surprised when a customer pointed out to us that he could employ zones and DTrace together to better understand and debug a three-tiered architecture by deploying the tiers together on a single host in separate zones in the test environment, and making specific queries using DTrace.

Experience

Zones is an integrated part of the Solaris 10 operating system, which is still under development. Through pre-release programs, Zones has seen adoption both within Sun and by a variety of customers.

In one "pilot" deployment, Sun's IT organization has consolidated a variety of business applications. A four-CPU server with six non-global zones is hosting:

- **Zone 1** The web front-end (Java System Web Server version 6.1) to Sun's host database.
- **Zone 2** The web front-end (Java System Web Server version 6.0) to the 'orgtool' website, providing Sun's online organization chart.
- **Zone 3** The Oracle database that provides the backend for Sun's online organization chart.
- **Zone 4** A database reporting tool, which interfaces with Peoplesoft and corporate tax databases; this is monitored by software from TeamQuest.
- **Zone 5** A security hardened CVS server, using LDAP and DNS name services (the other zones use NIS).
- **Zone 6** A Sun-internal application that utilizes Apache and MySQL.

This consolidation is probably typical of both large and small IT organizations; a wide variety of heterogeneous software (including different versions

```
# dtrace -n 'vminfo:::as_fault{@[zonename]=count()}'
dtrace: description 'vminfo:::as_fault' matched 1 probe
^C

  global                 4303
  lisa                   29867
```

Figure 8: Enhanced DTrace facility with zone context.

of the same application) is in play. In order to provide more predictable quality of service, the deployment team assigned different amounts of CPU shares to the various zones, to represent the relative importance of each workload.

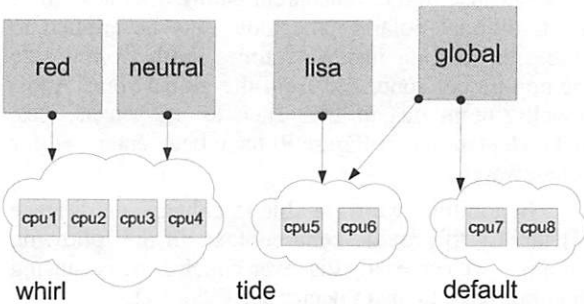


Figure 6: Zones and resource pools.

Security Experience

Any new systems architecture is rightfully viewed with suspicion by security conscious administrators; this was true during the project's development inside Sun. In order to better understand the security environment in which zones would need to operate, we created a non-global zone on an otherwise locked-down system. We then created a /SECRET file in the global zone and distributed the root password to the non-global zone far and wide within Sun, creating a "zones hacking" contest. This was extremely successful both for the contestants and the zones development team.

The system was compromised in the first few hours, using an exploit that we knew existed, but had considered very obscure. We realized that we had underestimated our adversaries. As we corrected the security problems our hackers found, we learned a lot about the sorts of attack techniques and vectors to expect. A positive result was that the reduced privileges associated with zone processes meant that attackers who managed to read the /SECRET file were usually unable to perform other sorts of mischief such as writing to /dev/kmem. We responded to the attacks by adding new system-level protections that prevent all of the exploits found. For example, mounts performed by a zone transparently have the nodevices mount option applied. This prevents using imported device files (for example, from an NFS share) as an attack vector.

Other Applications and Future Directions

In the course of developing this facility we considered the many other situations in which technologies such as Jails have been deployed. While the primary focus of the design is server consolidation, zones are well-suited for application developers, and may help organizations with large internal software development efforts to provide a multitude of "test systems." Many customers we have encountered spend substantial sums buying servers solely for this purpose.

Zones are also a useful solution for web hosting and other Internet-facing applications, in which creating a large number of application environments (perhaps administered by different departments) on modest hardware is important. We are also hopeful that advanced networking architectures such as PlanetLab will eventually include support for zones. At Sun, work is underway to prototype a version of Trusted Solaris based on the isolation provided by zones. We expect other novel uses for zones will emerge as researchers, developers, and administrators adopt them.

Moving forward, we know that networking poses key challenges to zones; groups of zones will cooperate in multi-tier architectures, and administrators will expect to be able to cluster, migrate, and failover zones from one host to another. Today these technologies are the unique domain of virtual machine solutions.

```
$ prstat -Z 10
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 12008 60028     191M  167M cpu18    1    0   0:00:31 1.1% ns-httpd/75
 28163 root       17M   10M sleep   59    0   4:40:37 0.5% ns-httpd/2
 12047 70002     296M  270M sleep   59    0   0:00:06 0.4% oracle/1
 10485 101       190M  101M sleep   59    0   1:37:20 0.2% webservd/82
 14058 root     6928K 5072K sleep   59    0   0:00:00 0.2% sshd/1
  1098 root     1736K  856K sleep   59    0   0:33:00 0.0% tqrtap.v9/1
   994 root     6848K 5512K sleep   59    0   0:23:08 0.0% tqwarp.ext/1
 12049 70002     296M  270M sleep    1    0   0:00:03 0.0% oracle/1
   804 root     4096K 3616K sleep   59    0   0:00:25 0.0% nscd/51
ZONEID    NPROC  SIZE   RSS MEMORY      TIME  CPU ZONE
     2       39  374M  272M   1.6%   4:45:01 2.1% lisa
     1       55 8025M 7217M    45%   0:05:20 0.9% red
     0       56  212M  130M   0.7%   2:28:18 0.2% global
     3       36  463M  211M   1.3%   1:48:55 0.2% neutral
     6       47  940M  372M   2.2%   0:24:52 0.0% euro
     5       38  330M  246M   1.5%   0:10:47 0.0% end
Total: 261 processes, 1356 lwps, load averages: 0.12, 0.13, 0.14
```

Figure 9: Monitoring Zones Using prstat. The top half of this split view shows the individual processes consuming the most CPU cycles. The bottom half shows a view of CPU usage aggregated by zone.

One of the strengths of zones is its integration with the base operating system. To provide a comprehensive solution, pervasive integration with the wider systems management software stack is necessary, and will be a major part of our future work.

Availability

Solaris Zones, which has been productized under the name *N1 Grid Containers*, is an integrated part of the Solaris 10 Operating System. Pre-release versions are available as part of the Software Express for Solaris Program at http://www.sun.com/solaris/10. A clearinghouse of information about Solaris Zones is available at http://www.sun.com/bigadmin/content/zones. Documentation is available at http://docs.sun.com.

Conclusions

A successful server consolidation must drive down both initial and recurring costs and day-to-day complexity for all involved. Having less hardware to manage is an important goal. However, the ability to maintain less software – fewer operating system instances – can have an even greater impact on the long-term cost reduction realized. The savings in operating system licenses and service contracts alone can be substantial. The best consolidations also allow a site to split the platform administration and application administration tasks. This capability allows the IT organization to delegate certain work responsibilities while maintaining control over the server itself, so areas of specialization can be exploited.

Solutions that create a hierarchy of control on a single host *without sacrificing observability* allow IT organizations to act as infrastructure providers who can provide compute resources, not just networks and SANs. Simultaneously, application expertise can remain with the department deploying or developing the application.

Solaris Zones offer the first fully realized facility for server consolidation built directly into a commodity operating system. Zones provides the namespace, security and resource isolation needed to drive effective consolidation in the real world.

Author Information

Daniel Price received a Bachelor of Science with honors in Computer Science from Brown University. At Brown, he got his first taste of UNIX systems administration serving as a SPOC (systems programmer, operator, consultant) for the Computer Science Department. He joined Sun Microsystems' Solaris Kernel Development Group in 1998 and has worked on several I/O frameworks, the Zones project, and on administering the OS group's development labs.

Andrew Tucker is a Distinguished Engineer in the Solaris Data and Kernel Services group at Sun Microsystems. He has been at Sun since 1994 working on a variety of projects related to the Solaris operating system, including scheduling, multiprocessor support, inter-process communication, clustering, resource management, and server virtualization. Most recently, he was the architect and technical lead for Solaris Zones. Andrew received a Ph.D. in Computer Science from Stanford University in 1994.

Acknowledgments

The Zones project was the work of a much larger group of engineers and staff than the authorship of this paper represents, and we are grateful to everyone who worked to make this project possible. For this paper, Allan Packer and Priya Sethuraman provided performance data, and Karen Chau and Norman Lyons provided information about the pilot deployment of zones in Sun's IT organization. In addition, a number of people contributed substantially to the writing of this paper. Penelope Cotten provided extensive editorial assistance. Dave Linder, John Beck, David Comay, Liane Praza, Bryan Cantrill and our USENIX shepherd, Snoopy, provided valuable feedback that made the paper much improved.

References

[1] Andrzejak, Artur, Martin Arlitt, and Jerry Rolia, *Bounding the Resource Savings of Utility Computing Models*, Technical Report HPL-2002-339, HP Labs, 2002.

[2] Barham, Paul, et al., "Xen and the Art of Virtualization," *Proceedings of the 19th Symposium on Operating Systems Principles*, 2003.

[3] Bavier, Andy, et al., "Operating system support for planetary-scale services," *Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[4] Cantrill, Bryan, Mike Shapiro, and Adam Leventhal, "Dynamic instrumentation of production systems," *USENIX Annual Technical Conference*, 2004.

[5] Charlesworth, Alan, et al., "The Starfire SMP interconnect," *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*, 1997.

[6] *How to Break Out of a chroot() Jail*, http://www.bpfh.net/simes/computing/chroot-break.html .

[7] Gum, P. H., "System/370 Extended Architecture: Facilities for Virtual Machines," *IBM Journal of Research and Development*, Vol. 27, Num. 6, 1983.

[8] IBM Corp., *Partitioning for the IBM eServer p-Series 690 system*, http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar.html .

[9] Kamp, Poul-Henning and Robert Watson, "Jails: Confining the Omnipotent Root," *Second International System Administration and Networking Conference (SANE 2000)*, May 2000.

[10] Sun Microsystems, Inc., *Solaris 9 Resource Manager Software*, http://wwws.sun.com/software/whitepapers/solaris9/srm.pdf .

[11] Sun Microsystems, Inc., *System Administration Guide: Security Services*, http://docs.sun.com/db/doc/816-4557 .

[12] Thibodeau, Patrick, "Data Center, Server Consolidations Remain Top IT Issue," *Computerworld*, March 2004.

[13] http://www.linux-vserver.org .

[14] Waldspurger, Carl, "Memory Resource Management in VMware ESX Server," *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, 2002.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

Member Benefits

- Free subscription to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and Open Source, book and software reviews, summaries of sessions at USENIX conferences, and Standards Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to *;login:* on the USENIX Web site.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, on the USENIX Web site.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as security, Linux, Internet technologies and systems, operating systems, and Windows—as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.
- Savings on a variety of products, books, software, and periodicals: see *http://www.usenix.org/membership/specialdisc.html* for details.

SAGE

SAGE is a Special Interest Group (SIG) of the USENIX Association. It is organized to advance the status of computer system administration as a profession, establish standards of professional excellence and recognize those who attain them, develop guidelines for improving the technical and managerial capabilities of members of the profession, and promote activities that advance the state of the art or the community.

USENIX & SAGE Thank Their Supporting Members

USENIX Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ Ajava Systems, Inc. ❖ AMD ❖
❖ Asian Development Bank ❖ Aptitune Corporation ❖ Atos Origin B.V. ❖
❖ Delmar Learning ❖ DoCoMo Communications Laboratories USA, Inc. ❖
❖ Electronic Frontier Foundation ❖ Hewlett-Packard ❖ IBM ❖ Interhack Corporation ❖
❖ MacConnection ❖ The Measurement Factory ❖ Oracle ❖ OSDL ❖
❖ Microsoft Research ❖ Perfect Order ❖ Portlock Software ❖ Raytheon ❖
❖ Sun Microsystems, Inc. ❖ Taos ❖ UUNET Technologies, Inc. ❖ Veritas Software ❖

SAGE Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ Ajava Systems, Inc. ❖ Asian Development Bank ❖
❖ FOTOSEARCH Stock Footage and Stock Photography ❖ Microsoft Research ❖
❖ MSB Associates ❖ Raytheon ❖ Ripe NCC ❖ Taos ❖

For more information about membership, conferences, or publications,
see *http://www.usenix.org/*
or contact:

USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA
Phone: 510-528-8649 Fax: 510-548-5738 Email: *office@usenix.org*

ISBN 1-931971-24-2